University of Zurich[UZH]

Department of Informatics

Crowd Process Design

How to coordinate crowds to solve complex problems

Dissertation submitted to the Faculty of Business,
Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor / Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Patrick M. de Boer
Citizen of Zug, Switzerland

approved in October 2017

at the request of
Prof. Dr. Abraham Bernstein
Prof. Dr. Philippe Cudré-Mauroux

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, October 25th, 2017

Chairman of the Doctoral Board: Prof. Dr. Sven Seuken

Abstract

The Internet facilitates an on-demand workforce, able to dynamically scale up and down depending on the requirements of a given project. Such *crowdsourcing* is increasingly used to engage workers available online. Similar to organizational design, where *business processes* are used to organize and coordinate employees, so-called *crowd processes* can be employed to facilitate work on a given problem. But as with business processes, it is unclear which crowd process performs best for a problem at hand. Aggravating the problem further, the impersonal, usually short-lived, relationship between an employer and crowd workers leads to major challenges in the organization of (crowd-) labor in general.

In this dissertation, we explore *crowd process design*. We start by finding a crowd process for a specific use case. We then outline a potential remedy for the more general problem of finding a crowd process for *any* use case.

The specific use case we focus on first, is an expert task, part of the review of statistical validity of research papers. Researchers often use statistical methods, such as t-test or ANOVA, to evaluate hypotheses. Recently, the use of such methods has been called into question. One of the reasons is that many studies fail to check the underlying *assumptions* of the employed statistical methods. This results in a threat to the statistical validity of a study and hampers the reuse of results. We propose an automated approach for checking the reporting of statistical assumptions. Our crowd process identifies reported assumptions in research papers achieving 85% accuracy.

Finding this crowd process took us more than a year, due to the trial-and-error approach underlying current crowd process design, where in some cases a candidate crowd process was not reliable enough, in some cases it was too expensive, and in others it took too long to complete. We address this issue in a more generic manner, through the automatic recombination of crowd processes for a given problem at hand based on an extensible repository of existing crowd process fragments. The potentially large number of candidate crowd processes derived for a given problem is subjected to Auto-Experimentation in order to identify a candidate matching a user's performance requirements. We implemented our approach as an Open Source system and called it PPLib (pronounced "People Lib"). PPLib is validated in two real-world experiments corresponding to two common crowdsourcing problems, where PPLib successfully identified crowd processes performing well for the respective problem domains.

In order to reduce the search cost for Auto-Experimentation, we then propose to use black-box optimization to identify a well-performing crowd process among a set of candidates. Specifically, we adopt Bayesian Optimization to approximate the maximum of a utility function quantifying the user's (business-) objectives while minimizing search cost. Our approach was implemented as an extension to PPLib and validated in a simulation and three real-world experiments.

Through an effective means to generate crowd process candidates for a given problem by recombination and by reducing the entry barriers to using black-box optimization for crowd process selection, PPLib has the potential to automate the tedious trial-and-error underlying the construction of a large share of today's crowd powered systems. Given the trends of an ever more connected future, where on-demand labor likely plays a key role, an efficient approach to organizing crowds is paramount. PPLib helps pave the way to an automated solution for this problem.

Zusammenfassung

Das Internet beherbergt on-demand Arbeitskräfte, die ermöglichen, ein Team sehr schnell dynamisch zu vergrössern/verkleinern abhängig von den Anforderungen an eine bestimmte Aufgabe. Dieses sogenannte *Crowdsourcing* wird immer häufiger für produktive Arbeit eingesetzt. Ähnlich wie im Feld der Organisationsgestaltung, wo Business Prozesse verwendet werden um Mitarbeiter zu organisieren und zu koordinieren, können in Crowdsourcing hierfür Crowd Prozesse eingesetzt werden. Jedoch gilt genau wie bei Business Prozessen, dass vor dessen Umsetzung unklar ist, welcher Crowd Prozess am besten für ein bestimmtes Problem geeignet ist. Komplizierter wird das ganze dadurch, dass die unpersönliche und häufig kurzlebige Beziehung zwischen einem Arbeitgeber und solchen Online Arbeitnehmern, genannt Crowd Worker, zu Schwierigkeiten bei der Organisation von Crowd-Arbeit führen kann. Diese Doktorarbeit befasst sich mit dem *Finden von geeigneten Crowd Prozessen.* Wir beginnen damit, für einen konkreten Anwendungsfall einen Crowd Prozess zu finden. Danach stellen wir eine mögliche Abhilfe für die allgemeinere Fragestellung vor: Das Finden eines Crowd Prozesses für *irgend ein* Problem.

Der konkrete Anwendungsfall, in dem wir für ein spezifisches Problem einen Crowd Prozess aufzeigen, gehört zum Review der statistischen Korrektheit eines Forschungsartikels. Forscher verwenden häufig statistische Methoden, wie zum Beispiel t-tests oder ANOVA, um Forschungshypothesen zu evaluieren. Solche statistische Methoden setzen jedoch gewisse Bedingungen ("*Assumptions*") voraus um angewendet werden zu können; Studien welche diese Assumptions vorher nicht in ihren Daten prüfen, gefährden die statistische Validität ihres Artikels und reduzieren die Verwendbarkeit der Studienresultate. Wir stellen einen automatisierten Ansatz vor, der prüft ob ein Forschungsartikel bestätigt, die notwendigen Assumptions getestet zu haben. Unser Crowd Prozess identifiziert solche Assumptions mit 85% Genauigkeit.

Um den Crowd Prozesses für diesen konkreten Anwendungsfall zu finden, benötigten wir mehr als ein Jahr. Hauptverantwortlich dafür ist der trial-and-error Ansatz, der bislang dem Crowd Prozess Design zugrunde liegt. Dabei ist manchmal ein Prozesskandidat nicht genügend verlässlich, manchmal zu teuer und in wieder anderen Fällen zu langsam. Wir befassten uns deswegen mit einer generischen Lösung für Crowd Prozess Design, mittels automatischer Rekombination von Crowd Prozessen für ein gegebenes Problem basierend auf einer Sammlung an existierenden Fragmenten von bekannten Crowd Prozessen. Die potenziell grosse Anzahl Kandidatenprozesse, die für ein bestimmtes Problem mittels Rekombination abgeleitet werden kann, wird durch Auto-Experimentation gefiltert, um einen Kandidaten zu nominieren, der den Anforderungen eines User's entspricht. Wir implementierten unseren Ansatz als Open Source System und nannten es PPLib (ausgesprochen "People Lib"). PPLib wird in zwei typsichen Crowd Prozess Problemstellungen evaluiert.

Um die Suchkosten von Auto-Experimentation zu verringern, stellen wir danach ein Black-Box Optimierungsverfahren vor, das performante Crowd Prozesse effizient in einer Liste von Kandidaten identifiziert. Nach einer Quantifizierung der Bedürfnisse des Benutzers in einer Nutzenfunktion, können wir Bayesische Optimierung verwenden, um deren Maximum effizient zu approximieren. Unser Ansatz wurde als Erweiterung zu PPLib implementiert und sowohl in einer Simulation als auch in drei realen Anwendungsfällen evaluiert.

Durch einen effektiven Ansatz zur Generierung von Crowd Prozessen für ein gegebenes Problem mittels Rekombination, sowie auch durch die Herabsetzung der Barrieren für Black-Box Optimierung für Crowd Prozess Selektion, hat PPLib das Potenzial das langwierige trial-and-error bei der Erstellungen von Crowdsourcing Systemen zu automatisieren. Es ist zu erwarten, dass on-demand Arbeit in Zukunft eine wichtige Rolle spielen wird. Eine effiziente Methode, um Crowds zu organisieren, ist hierfür eine Grundvoraussetzung. PPLib hilft, den Weg dafür zu ebnen.

Acknowledgements

"Live as if you were to die tomorrow. Learn as if you were to live forever." – Mahatma Gandhi

I devoted the past few years to the path of knowledge, hoping to satisfy my curiosity. This hope was quickly downgraded to the hope of at least keeping my curiosity in check. Mostly. A PhD turns out to be much more than the consumption, processing and creation of knowledge; it sometimes leads to reflection about the nature of knowledge itself. While on this path, I realized an important metaphor that would help me keep my curiosity in check: knowledge can be related to a (dense) graph; where learning would then constitute a graph traversal problem. Since the number of discovered nodes in graph traversal is (usually) exponentially more than the number of visited nodes, the more I learnt, the more I learnt about what I do not know yet – hence fueling curiosity further. A vicious cycle – but with the very positive side-effect of increasing knowledge.

I am especially grateful to my mentor and advisor, Avi, whose heuristics helped guide my traversal of the knowledge graph. His mental flexibility, speed and creativity make him a great mentor; his communicative skills a superb discussion partner and his openness a good friend.

Thank you Mike for helping to make the office a fun place from much more than just a visual perspective. Besides the many shared laughs, I'm grateful for the countless insightful discussions about science, politics and ideas, and the impact they had on me. I would also like to thank the members of my research group for their support and for helping me see more vantage points: Bibek, Cosmin, Daniel, Daniele, Ela, Lorenz, Matthias, Philipp, Shen and Tobi. Thank you to the people of the Canton of Zurich, the University of Zurich, the Department for Informatics and the Swiss National fund for giving me the opportunity to study in this amazing environment.

Finally, I would like to thank my family and friends for supporting me on my journey, for sharing the ups and downs through the haze of uncertainty in the research process. My parents and sister, for building a ship that would carry me and my friends for helping me navigate. Most importantly, the greatest source of inspiration and support has been my love and best friend, Andrea. Thank you!

Table of Contents

Part I, Synopsis

Synopsis

1 Introduction

The Division of Labor was famously described by Adam Smith in *The Wealth of Nations, 1776*, and became a major driver of technological development after the Industrial Revolution. By splitting problems into sub problems, people could specialize on one aspect of an overall problem and coordinate to integrate their solutions. Such specialization enables individual workers to become more efficient for subtasks falling into their designated fields. This leads to large productivity gains in solving complex problems and enabled humanity to build the airplanes, skyscrapers and smartphones that are part of our lives today.

[Malone et al. 2011] argue, that specialization deepened with the advent of the internet and the ever increasing number of people contributing to the content available online. The internet massively reduced the cost of communication among people and fostered much wider distribution of labor than ever before – be it voluntary labor or paid. Numerous examples showcase the potential of well-directed efforts of massive amounts of people, such as Wikipedia, Folding@Home (leading to advances in the complex topic of protein folding), GalaxyZoo (leading to the classification of hundreds of thousands of galaxies), OpenStreetMap or reCaptcha (leading to the digitization of books). Due to the large number of people involved, such efforts have been termed *crowdsourcing*.

Organizations have started to tap into the creative potential of the crowd through initiatives like open innovation contests, a coordination structure, where crowds compete against each other in contests centered around a given topic. For example, Roche, a large pharma company, sponsored a contest on InnoCentive[1] in 2008 to find a better way for measuring the volume and quality of clinical specimen passing through its automated chemistry analyzers [Malone et al. 2011]. Within two months, the crowd identified a novel solution that eluded Roche for 15 years. In another example taking place after the oil spill of the sinking DeepWater Horizon platform in 2010, BP turned to crowdsourcing to find more effective ways to remove oil from the ocean [Nagar et al. 2016]. A local fisherman proposed a method that turned out to be more effective per vessel than the entire BP fleet at the time. However, establishing an open innovation contest for a given problem is difficult, since a crowd with the sought knowledge first needs to be recruited and properly incentivized to participate.

As an alternative, online labor markets (OLM) emerged, where crowd workers offer their services for hire. In 2017, there exist a myriad of OLM specializing in different types of tasks: for example, micro tasks (e.g. Amazon Mechanical Turk[2], CrowdFlower[3]), general-purpose tasks (e.g. UpWork[4],

[1] https://www.innocentive.com/
[2] http://mturk.com
[3] https://www.crowdflower.com/
[4] https://www.upwork.com/

Freelancer[5]) or Creative tasks (e.g. Fiverr[6]). Reminiscent of the principles behind cloud computing, OLM provide crowdsourcing *requesters* the opportunity to dynamically form a workforce for a given problem, as well as scaling it up and down based on the needs of a job. Such properties can be very advantageous for requesters. For example, it can be effectively leveraged in scenarios where job demands are irregular and sometimes many people are needed and sometimes few. In some OLM, requesters can hire crowd workers based on their attributes, which allows targeted hiring from pools of specialists made up of people all around the world. Examples of such attributes include certifications and knowledge domains with experience level (e.g. 5 years of experience in Java programming).

The first generation of OLM's offer insight into the large potential of crowdsourcing, but there were many criticisms and limitations to its application. For example, the highly varying quality of crowdwork lead to the emergence of the active field of quality assurance in crowdsourcing. [Ipeirotis 2010] notes an overt focus on trivial work, such as transcription, object classification and data entry. Additionally, [Kocsis and de Vreede 2016] point out ethical considerations behind crowdsourcing. [Kittur et al. 2013] therefore identified opportunities for improvement of this first generation, which paved the way for OLM's to mature. In 2017, a transition to more high-level work is visible. For example, UpWork, an expert crowdsourcing platform, report that large fraction of their workers are highly qualified, as indicated by [Popiel 2017], who found 77% of registered workers to have at least a college degree. UpWork is considered one of the large players in the field, they paid upwards of 1 billion USD in salaries in 2016.

The growing number of people of varying backgrounds participating in crowdsourcing systems (such as OLM's), allow for an increasing *distribution of labor* and *specialization* in problem solving. Complex crowdsourcing problems can be broken into more specific sub problems that a specialized crowd can work on solving. As more and more sub problems need to be addressed, the challenge becomes coordinating across sub problems; we therefore asked a central question about the division of labor:

How can crowds be coordinated effectively to solve complex problems?

[5] https://www.freelancer.com/

[6] https://www.fiverr.com/

2 Problem statement

Finding a suitable coordination structure for a given problem is not unique to crowdsourcing. In fact, it is a key area of research in the field of Coordination Science, which is studied in many different disciplines [Malone and Crowston 1994], including computer science (e.g. distributed computing), political science (e.g. reorientation of economic systems after industrial revolution), management science (e.g. organizational design), animal behavior (e.g. self-organization of foraging bees) and psychology (e.g. team formation).

The remote, on-demand nature of crowdsourcing, however, means that this approach comes with its own unique set of coordination challenges. For example, crowdsourcing is based on (often anonymous) people collaborating remotely, which means that a personal connection between employer and employee is missing. This may lead to lower reliability and lower quality of work. The missing personal connection also makes teamwork harder, since people might feel less accountable to each other [Salehi et al. 2016]. These systemic difficulties in communication make the division of labor in crowdsourcing more challenging. More specifically, a common problem when using crowdsourcing is how to coordinate work on the various sub problems, i.e. the coordination structure of the people involved.

Existing research in crowdsourcing offers a wealth of approaches containing creative solutions with coordination structures for many individual problems; but few researchers looked at the more general question of *finding a suitable coordination structure for <u>any given</u> crowdsourcing problem*. Probably one of the closest topics in crowdsourcing research to this question are design patterns for crowdsourcing workflows, similar to the ones available in Software Engineering: e.g. Find-Fix-Verify [Bernstein et al. 2010a], Iterative Refinement [Little 2010] and Context Trees [Verroios and Bernstein 2014]. The key problem with design patterns, though, is that their performance for a given problem often only becomes clear after trying them. In other words, it is not clear ex-ante which design pattern would perform particularly well for a given problem.

The first research question of this dissertation will revolve around a case study, where we faced exactly this problem. In a painstaking trial-and-error process, we implemented and tried many different design patterns and coordination structures adapted to our case, until we found one with acceptable performance characteristics for our specific problem. It took more than a year of trial-and-error until we succeeded. The difficulty of finding a coordination structure isn't limited to our specific case; the existence of the plethora of published approaches and systems for specific problems gives a clue that other crowd process designers have had similar experiences. Aggravating the problem further, as we would learn on our journey to a possible remedy of this issue, the performance of individual coordination structures varies widely for different problems. In other words: the same crowd process would perform differently in different problem contexts. This leads to the insight, that just picking one crowd process without trying others potentially carries large opportunity cost.

3 Terminology

Before proceeding to the Research Questions, Table 1 defines the terms used in the remainder of this dissertation.

Coordination structure	[Malone 1987] defines a coordination structure as *"a pattern of decision-making and communication among a set of actors who perform tasks in order to achieve goals"*.
(Crowd-) Workflow	We use the term crowd workflow as an enactment of a coordination structure in crowdsourcing.

(Crowd-) Process	We use the terms *Crowd Workflow* and *Crowd Process* synonymously.
Crowd Process Design	Refers to the creation of a crowd workflow for a problem a requestor/user would like to solve through crowdsourcing.
(Crowd Process-) Fitness	The degree to which the performance of a crowd process satisfies a user's requirements for a given problem. We use the terms (Crowd Process) fitness and utility interchangeably.

Table 1: Terminology

4 Research Questions and Hypotheses

We explored the topic of crowd coordination in 3 distinct steps translating into their respective research questions:

1. First, motivating the need for improvements to the art of *crowd process design*, we report on a case study, where we attempted to find a well-performing coordination structure for (non-expert) crowds in a typical expert topic
2. Second, we explore a methodology for automatically deriving crowd processes for a given problem.
3. Third, we explore a research question focused on picking a crowd process among a potentially large set of generated candidates fitting a given problem

For our first research question, we step into the shoes of an employer looking to crowdsource an expert-task, which should be processed at a large scale. Such expert-tasks are often highly unstructured and require tacit knowledge to be completed successfully. This kind of crowdsourcing is a common way to reduce cost for expert-tasks, and usually involves engaging a large number of non-experts that are coordinated effectively to reach results comparable with experts. Therefore, the main difficulty for these is that due to the scarcity of experts, expert-tasks cannot be processed at scale. In order to scale well, a large worker pool needs to be used; which for cost reasons is usually a pool of non-experts. This implies a reduction of complexity by finding more structured representations of a task and circumvents the requirement for tacit knowledge by either making it explicit or avoiding it altogether. Once the task complexity has been reduced in this manner, a coordination structure needs to be identified that supports non-expert workers and assures the quality of their output.

The expert task we focus on in this case study is situated within *the review process of statistics of research articles*. Research articles are typically peer-reviewed before being published at a venue; a specific focus of such peer-reviews lies on the validation of the statistics typically used to back an article's claims. Since formal training in statistics is not a common attribute in the general population – and hence neither in large crowd worker pools – the main difficulty in finding a crowd process for this problem lies in reducing the task's dependence on knowledge of statistics, as well as finding a reliable coordination structure to assure consistent quality.

Q1: How can crowds identify whether statistical assumptions were reported for statistical methods applied in a research article?

When reviewing research articles, many research disciplines emphasize the need for internal and external validity of the research. A very common way to attribute both is using statistics, with null-hypothesis significance testing (NHST) still being very popular, despite growing criticism [Nuzzo 2014; Dragicevic 2016]. For example, to assess the difference between two conditions using a t-test (a commonly used *statistical method*), one first needs to ascertain, that the data (at least approximately) follows the normal distribution [Srivastava 2016; Hoekstra et al. 2012] (i.e., one needs to check what we will call a *statistical assumption*). Even if some methods are robust to violations of their assumptions in certain cases, it is essential to report on these violations [Good and Hardin 2012], because the degree of violation and the interactions between different underlying assumptions can undermine the validity of the results and hereby may hamper the reproducibility of the research. [7]

A lack of reporting of statistical assumptions therefore leaves a reviewer and future readers of an article in the dark as to the statistical validity of the article. Essentially, in that case, (peer-) reviewers are asked to blindly trust authors; a situation that stands in contrast to the mission of peer review and to the scientific method at large. It would therefore be interesting to shed light on the current state of assumption reporting in our community, CHI. To do so, we need to conduct a review of a representative sample of published CHI papers.

In this research question, we look for an automated way using crowdsourcing to assess whether statistical assumptions for a given statistical method have been reported in an article or not. Applying crowdsourcing to this problem would allow assessing this aspect on a large scale at low cost, which enables investigating the reporting standards of a scientific publishing venue over time. The main difficulty in doing so lies in enabling people to work on this question without them requiring any knowledge of statistics. Therefore, a coordination structure is required, that facilitates this prerequisite and guarantees high quality despite absence of knowledge in statistics.

H1: There exists a coordination method that allows non-expert crowd workers to identify reported statistical assumptions for statistical methods used in research reports.

We explore Hypothesis H1 with a candidate solution to this problem, a blend of pattern matching and crowdsourcing. In a first step, all synonyms of assumptions expected for a specific statistical method applied in an article are matched and extracted. Crowd workers could then evaluate each individual match of an assumption for a given method (i.e., a method-assumption pair) in the text, and check if the match indeed represents an actual reporting of the assumption under analysis (more on page 23).

This hypothesis can be evaluated like a binary classification problem, where for each research paper in a corpus, we check whether our application finds and correctly classifies all expected method-assumption pairs. Errors can be expressed through false-positives and false-negatives. To incorporate both types of errors into a performance metric evaluating the degree to which this hypothesis is satisfied, *Precision, Recall* and the *F1-Measure* may be used.

This crowd process took us more than one year to find. We tried many different coordination structures – but most of them didn't deliver the desired performance characteristics. Working on this case study convinced us of the gravity of a bigger problem behind the organization of crowd work: *coming up with*

[7] This paragraph was cited from the first article in this dissertation, see page 22

coordination structures that might work is very difficult. The main remedy to this problem currently are (crowd process) design patterns – but in this case study, some common patterns failed to work altogether, while others' performance was unreliable over multiple executions. Essentially, it is unclear how a specific pattern performs before trying it. To try it, we then had to implement the design pattern for our particular problem. This cost a lot of time, which motivated us to find an automatic solution for the manual work involved in generating ideas for different crowd processes and evaluating them. In the following, we therefore explore, how to *automatically derive candidate crowd processes for a given problem* and then *how to select one fitting the users performance characteristics*.

Q2: How can we automatically derive a set of candidate crowd processes for a given problem?

At ICIS'1999, [Bernstein et al. 1999] proposed the Process Recombinator; a tool to generate new business process ideas by recombining elements from a business process repository. Borrowing from object-oriented programming languages, the Process Recombinator views sequences of business processes as abstract elements that can be specialized into concrete outcomes. This idea can set the baseline for our own exploration into crowd process recombination.

H2.1: Typical crowdsourcing problems can be specified in terms of their abstract operators and their problem constraints.

CrowdLang [Minder and Bernstein 2012], a precursor to the work developed in this dissertation, uses a workflow-based language for problem specification. We see this as first evidence, that a problem specification through abstract operators is possible. PPLib is not a workflow-based language, hence, it uses a different set of operators than CrowdLang. We will therefore need to show that a problem specification through the use of its operators is possible.

We will test this hypothesis by illustration. More specifically, we show that multiple popular crowdsourcing problems can be specified in terms of their abstract operators in PPLib.

H2.2: A problem specification in terms of abstract operators can be used to generate many distinct crowd processes given a repository of corresponding crowd process fragments

This hypothesis explores the usefulness of the adoption of abstract operators for problem specification. It only makes sense to specify a problem this way, if this specification can be used as a starting point to derive many candidate solutions.

The hypothesis can be tested by illustration, where we show the number of crowd processes derived for multiple popular crowdsourcing problems.

Q3: How can we identify a crowd process that satisfies a user's performance requirements for a given problem among a set of candidate processes?

Through Q2, we have explored a means to generate (many) candidate crowd processes for a problem at hand. Different candidate crowd processes might vary in their degree of compliance with a user's performance requirements. Therefore, an efficient method is needed, to pick a crowd process among a set of candidates that fits a user's performance requirements well.

This formulation implies, that each crowd process's fitness with respect to a user's requirements needs to be quantified. In many cases, *cost* and *duration* will play a central role for a user, but different users might require different tradeoffs between the two and some require additional quality metrics for *task adequacy*. Our approach therefore leaves the definition of the fitness function and the relative weights of the incorporated metrics to the user.

Following our observation in the case study (see Q1 on page 12), we could not predict the performance of a crowd process accurately before running it. Therefore, a safer way to evaluate crowd process fitness is to run the crowd process in a real-world setting and quantify the desired fitness metrics afterwards: *Auto-Experimentation*. It is important to note, that this strategy only pays off, if a crowd process is sought, that will be applied in a larger use case. For example, if the goal of a crowdsourcing project is to translate a book, one could take a sample, e.g. 1 page of the book, and run a crowd process candidate on that sample in a real-world setting (e.g. 3 bilingual crowd workers editing the sample page concurrently in a Word processor for 1hr). Calculating crowd process fitness a posteriori instead of a priori avoids the problem of estimation accuracy. It does, however, introduce new problems related to the cost of calculating process fitness: Calculating fitness for many crowd processes in this manner may get costly.

Another challenge when quantifying crowd process fitness through Auto-Experimentation is the variability of the crowd workers participating in a process. More specifically, the same crowd process – if executed twice – might generate two completely different fitness values, if people with opposite skill levels are employed (and if the worker skill level of a crowd process can't accurately be controlled for due to limited data, e.g. in Amazon Mechanical Turk). Hence, over multiple executions, crowd process fitness is typically rather noisy. This could be treated as crowd process non-determinism; and a crowd process selection mechanism should take it into account when estimating process fitness.

A simple way to account for non-determinism is by re-executing the same process multiple times. We call this strategy Naïve Auto-Experimentation (NAE). In the following, we will refer to it as NAE-*K*, where each process in a set of candidates would be executed *K*-times and its results evaluated through a user-defined utility function. The optimal value of *K* is subject to a tradeoff between reliability and cost; whereas it might be hard to estimate a suitable value for *K* for processes whose fitness-distribution is of exponential nature (e.g. a long-tail distribution, with most results being ok, but few being outrageously wrong).

A more efficient approach to crowd process selection could involve the use of mathematical optimization; i.e. finding the crowd process with maximal fitness. An optimization technique would need to overcome three challenges to be useful:

- It would need to be applicable to a case where the closed-form expression of the objective function is not available. In the vast majority of cases, the mathematical representation of an objective function for crowd process fitness of a set of crowd processes is not available (a priori).
- It would need to be effective even when facing expensive objective functions. In case of crowd process fitness, the objective function involves actually running the crowd process on a sample, which can get costly if many processes need to be evaluated.

- It would need to be able to incorporate crowd process non-determinism. If the same crowd process, called with equal parameters, returns results with a high variance, an optimizer might easily be lead astray.

If the distribution of crowd process fitness is approximately normal, Bayesian Optimization [Brochu et al. 2010] may overcome these challenges. We therefore explored the usefulness of Bayesian Optimization for Auto-Experimentation in crowd process selection (BOA).

H3: When used for crowd process selection among a set of candidates, Bayesian Optimized Auto-Execution (BOA) nominates suitable candidates.

H3 can be divided into two sub hypotheses:

- H3.1: BOA nominates processes of equal or higher quality than have been proposed by experts before for a given problem

- H3.2: BOA finds good processes at a lower price than NAE

H3.2 mentions 'good' as quality baseline when comparing cost between NAE and BOA. This can be operationalized by reusing the definition of H3.1: *processes of equal or higher quality than have been proposed by experts before for a given problem*. Hence, both sub hypotheses need BOA to run real-world use cases for crowd process selection where experts have proposed solutions before and BOA's nomination can be compared against. Both sub hypotheses assume at least one expert-proposed process to be among the set of crowd process candidates used as input to BOA.

Furthermore, a comparison between NAE and BOA in terms of result quality would be interesting, since NAE can be seen as the current industry standard of evaluating crowd processes. In other words: *Are processes nominated by BOA of comparable quality as NAE, the current gold standard?* Unfortunately, it would be very difficult to operationalize a corresponding hypothesis due to the requirement for nominated processes to be of *comparable* quality. Checking this hypothesis entails the use of a crowd process quality metric, as well as the definition of some tolerance value. The definition for a tolerance would need to be based on third party literature for it to be effective and trustworthy in this case; however, there currently exists no universal definition for the context of crowd process fitness. We therefore refrain from validating a corresponding hypothesis, and instead quantify *the absolute difference in crowd process quality* for the nominated processes of the two approaches. We use *crowd process fitness* as the underlying metric for the comparison.

5 Contribution Summaries

The research questions and their hypotheses were explored in 3 distinct research projects. This section summarizes their findings and relates them to the proposed research questions.

The key contributions (see Figure 1) of this dissertation, follow the research questions outlined in the preceding section:

- Q1 (Case study for crowd process design): A scalable method and system to check the reporting of statistical assumptions in research papers and an analysis of the reporting standards in a popular HCI conference (CHI).
- Q2: A Method and System to derive crowd processes for a given problem

- Q3: A Method and System to efficiently select crowd processes among hundreds of candidates

Figure 1: Relationships between the key contributions of this dissertation

All three projects were concerned with finding suitable coordination structures in crowd sourcing, building up on each other (see Figure 1). The first project describes a case study for finding a coordination structure for one specific problem. This case study inspired the second project, which outlines and tests a more generic solution that could apply to any crowdsourcing problem. The proposed solution in the second project would be prohibitively expensive in practice, however. We therefore focus on reducing its cost in the third project.

5.1 Q1: Case Study for crowd process design

The first research project represents a case study in crowd process design, where we describe a novel crowd process to let non-experts review the reporting of statistical assumption in research papers. The crowd process is evaluated by comparing its results with ground truth established by experts. More specifically, we obtained N=100 research papers (power > 90%) published in a top-tier HCI conference (CHI), and found our crowd process to identify reported assumptions with a Precision of 0.82 and a

Recall of 0.83. Essentially, this means that 82% of assumptions returned by the crowd process were also reported according to the ground truth, and that 83% of the assumptions in the ground truth were also identified by the crowd process. Given that the inter-rater agreement (Fleiss-Kappa) among the 3 annotators who created the ground truth was 0.79, a Precision and Recall around this number can be considered similar to human performance. These results support H1, which suggests that non-expert crowd workers are able to identify reported statistical assumptions for statistical methods used in research reports.

We then used our crowd process to analyze the reporting standards of statistical assumption in a top-tier HCI conference and find that only very few authors of published papers report checking their statistical assumptions, with little evidence for improvement over time.

5.2 Q2: Towards automated crowd process design

The second research project aims to improve one aspect of crowd process design: The problem of coming up with multiple candidates for crowd processes that might work well for the crowdsourcing problem at hand. It describes and evaluates a method and programming framework called *PPLib,* which uses a formulation of a crowdsourcing problem in terms of abstract operators to recombine candidate solutions from an extensible crowd process repository. Through the use of Auto-Experimentation, i.e. by actually executing the process candidates on a sample of the crowdsourcing problem, the best alternative is established. The framework is evaluated by applying it to 2 real-world crowdsourcing problems, where it successfully generated at least 5 suitable crowd processes for each. More specifically, Hypothesis 2.1 is supported by the formalization of the two real-world crowdsourcing problems through abstract operators, and Hypothesis 2.2 is supported by the fact, that 41 candidate processes could be derived for each problem, of which at least 5 satisfied the users requirements in each case.

5.3 Q3: Improving the efficiency of automated crowd process design

The third research project focusses on the identification of a crowd process fitting a user's requirements among a potentially large set of candidate crowd processes. This problem is a direct result out of the second research project, where a large list of crowd process candidates may be generated for a given crowdsourcing problem. Naively executing a large list of candidates - the approach used in the second project, called NAE - would be prohibitively expensive in a real-world scenario. Therefore, we propose to turn crowd process selection into a black-box optimization problem and implemented a variant of Bayesian Optimization as a module of PPLib, called BOA, to solve it. It is evaluated in one simulation and two real-world scenarios. In the simulation, BOA is compared to NAE, by quantifying the difference in crowd process fitness of the nominated crowd process of each respective strategy (BOA vs NAE). We found BOA to nominate slightly less well performing processes (according to process fitness) than the most precise setting of NAE (a 12.5% difference of fitness) at 50x lower cost. However, at the lowest cost setting of NAE (still 17x higher cost than BOA), BOA performed 35% better (again in terms of fitness).

To investigate H3.1 and H3.2, we then executed BOA for a well-known real-world crowdsourcing problem, for which expert-designed solutions existed. BOA nominated the 2nd best process of the overall 41 candidates at roughly 10x lower cost than NAE. The nominated process was a variant of a popular expert-designed solution. This finding supports H3.1 for BOA's nomination of an expert-proposed process and H3.2 due to the 10x cost advantage of BOA.

In the second real-world evaluation, we executed BOA on another well-known problem in crowdsourcing, where, in combination with our framework, it nominated a crowd process that

performed better than the gold standard at the time. This finding supports H3.1. Additionally, we calculate the lower cost bound of NAE and find it to be 10x higher than the money spent by BOA, which supports H3.2. Moreover, the formulation of an additional common crowdsourcing problem in terms of its abstract operators further supports Hypothesis 2.1.

6 Outline

In the following, Part II presents the three articles this cumulative dissertation is comprised of. First, we present a case study in crowd process design (page 23), followed by a description of a methodology to derive crowd processes for any crowdsourcing problem (page 39). The last article details a means to make the selection of a fitting crowd process among a set of candidates more efficient (page 59).

Part III outlines the limitations of the reported findings and presents directions for future work, followed by a conclusion.

Part II, Papers

Assessing Statistical Assumption Reporting in CHI

An extension of this chapter is currently to be submitted ToCHI. The authors contributing to this chapter are: Patrick M. de Boer, Michael Feldman, Manuel Roesch, Jen Mankoff (Carnegie Mellon University), Abraham Bernstein.

Assessing Statistical Assumption Reporting in CHI

ABSTRACT

Researchers often use statistical methods, such as t-test or ANOVA, to evaluate hypotheses. Recently, the use of such methods has been called into question. One of the reasons is, that many studies fail to check the underlying assumptions of the employed statistical methods. This results in a threat to the statistical validity of a study and hampers the reuse of results. We propose an automated approach for checking the reporting of statistical assumptions. Our approach coordinates crowd workers in identifying reported assumptions in research papers achieving 85% accuracy. When applied to a sample of 261 CHI papers published between 1989 and 2016, our analysis found that only 13% of papers using ANOVA or t-test reported at least one of their assumptions. We therefore hope that approaches like ours will help researchers and reviewers to improve the statistical reporting in scientific publications, potentially contributing to a greater confidence in the results.

1 Introduction

Recently, researchers across disciplines have become concerned about the abuse of null-hypothesis statistical testing (NHST) methods [Nuzzo 2014; Dragicevic 2016]. This discussion surfaced in CHI, for example, in the form of ignoring effect sizes [Kaptein and Robertson 2012].

Statistical validity depends on a large set of factors; in this paper, we are primarily concerned with statistical assumptions. For example, to assess the difference between two conditions using a t-test (a commonly used *statistical method*), one first needs to ensure, that the data (at least approximately) follows normal distribution [Srivastava 2016; Hoekstra et al. 2012] (i.e., one needs to check what we will call a *statistical assumption*). Even if some methods are robust to violations of their assumptions in certain cases, it is essential to report on these violations [Good and Hardin 2012]. Depending on the degree of the violation, the validity of the results may be undermined, which may reduce the reproducibility of the research. Conversely, lack of reporting of assumptions could mean that some assumptions were never tested, which therefore raises questions about the validity of the *statistical methods* being used. Lack of reporting could also mean that the statistical assumptions are tested and not described. Both conditions contributing to our status-quo, where reviewers are asked to almost blindly trust authors on their data analysis – a culture that stands in contrast to the scientific method.

However, raising the standard for reporting is difficult. The current review process does not emphasize assumption reporting, perhaps because limited reviewer time is needed for larger issues in most papers. As a result, editors lack a consistent, easy way of measuring whether statistical assumptions are reported, and thus it is difficult to set standards regarding this issue.

Thus, we argue that an automated approach for testing whether publications meet basic assumption reporting standards would (1) make it easier to track reporting quality over time, (2) provide authors with a simple way to check whether they are meeting a common standard, and (3) provide editors with a simple way to flag potentially problematic papers for further investigation. However, assessing even basic issues like the presence or absence of statements about whether data meets expected statistical assumptions requires reading scientific writing, a challenging problem on its own.

Our primary contribution is a method for assessing the presence or absence of statistical assumption reporting in published works. More specifically, we use an extensible rule-base to determine the assumptions expected to be reported for the applied statistical methods. Methods with non-reported assumptions are flagged for further review. Our approach is based on crowdsourcing. We employ crowd workers to validate the presence of assumptions in a paper. We do not require workers to have

any prior education in statistics, since tasks are structured as a sequence of simple questions on text semantics combined with algorithms to aggregate crowd answers. Crowds have been shown to perform as well as (or even better than) experts in many cases [Surowiecki 2005; Introne et al. 2011; Lintott et al. 2008]; employing them for the prescreening of research papers could save experts time and help uphold a common standard.

We validate our approach on a dataset of 100 papers published in CHI's proceedings between 1989 and 2016 (*validation*-dataset). We manually established ground truth for all of these papers, and show that our crowdsourced approach is able to achieve 85% accuracy, 82% precision, and 83% recall (Cohen's kappa =0.70), which is similar to the inter-rater agreement on the *validation*-dataset (Fleiss' Kappa=0.79). Note, that our system is designed to serve as a tool to assist reviewers and/or authors, not to completely replace reviewers with crowds.

Our second contribution is an analysis of the reporting of statistical assumptions in CHI over time, where we sampled a total of 261 papers over the years (*chi*-dataset). Our data indicates that most papers in CHI using at least one of two common statistical methods (t-test or ANOVA) do not report any assumptions (87%). Additionally, we find evidence for mild improvement of the fraction of papers reporting at least one assumption over time.

The correct approaches to hypothesis testing, as well as the correct standard for reporting, are subjects of a broader and yet to be resolved conversation. We demonstrate that standards, once set, could potentially be supported in a transparent and accessible fashion that could benefit both, authors and reviewers. Our work also calls into question whether we as a community have been meeting our own implicit standards or those of the fields that we draw on methodologically.

2 Related Work

A large fraction of published studies contain at least statistical flaws or, at worst, statistical errors [Strasak et al. 2007]. The problems can be roughly classified into being related to sample size and power calculation (*e.g.,* [Kuzon et al. 1997]), missing reporting, so-called 'p hacking' [Baker 2016; Leek and Peng 2015], inference of causality from a correlation, confounding variables, choice of statistical method, adjustments for multiple hypothesis testing [Aiken et al. 1999] and breaking statistical assumptions [Rouder et al. 2016; Westfall and Yarkoni 2016].

One of the most common and most criticized statistical concepts, null hypothesis statistical testing or NHST, drew particular attention in the recent years due to the gravity of its shortcomings [Ranstam 2012; Nuzzo 2014], with some journals even banning its use. [8] Some argue that NHST was originally intended to be used as evidence that a hypothesis is worthy of a second look, but not a stand-alone statistical tool [Nuzzo 2014].

The main problems [Kaptein and Robertson 2012] of the current practice are (i) the fallacy of the transposed conditional (*i.e.,* "Absence of evidence is not evidence of absence"), (ii) a neglect of statistical power, and (iii) the lack of attention to effect size. Potential solutions to these problems include paying close attention to effect size and confidence intervals [Dragicevic 2016]. Alternatively, other statistical approaches than NHST could be used. For example, *Bayesian Statistics* [Kay et al. 2016] are more precise with small-sample studies and thus arguably a better fit for the style of research common in CHI. Another proposed alternative to NHST is *Magnitude-based Inference* [Van Schaik and Weston 2016]. Differently from NHST, the focus here is not on understanding whether a treatment makes a difference, but rather on estimating how likely it is that a treatment's difference will be big enough to be considered

[8] https://www.sciencenews.org/blog/context/p-value-ban-small-step-journal-giant-leap-science

important. Despite the existence of such alternatives, scientific research at large and CHI in particular are still mostly wedded to NHST. Given how commonly NHST is used (and its weaknesses), it is critical to ensure that it is used as correctly as possible.

2.1 Statistical Assumptions as one of the pitfalls in NHST

At its heart, NHST is a mathematical technique that depends on specific mathematical assumptions. These assumptions have to be satisfied at least approximately, in order to reach meaningful and valid results [Good and Hardin 2012; Veldkamp et al. 2014]. Despite the importance of assumption testing, Hoekstra *et al*. [Hoekstra et al. 2012] show that the underlying assumptions of common methods, such as t-tests, are often not evaluated and/or overlooked while reporting the methodology of data analysis.

One prominent example of a rarely tested assumption is homoscedasticity (the variance of a dependent variable across the range of values of the corresponding independent variable). Techniques in the correlation family (Pearson's correlation, linear regression, *etc.*) assume similar variance across values (high homoscedasticity) (*e.g.,* see [Osborne and Waters 2002]). A violation of this assumption can lead to substantial distortion of the outcome and seriously weaken the results by increasing the chance of a Type I error [Berry and Feldman 1985]. Note, that various terms are used interchangeably for homoscedasticity, e.g. homogeneity of variance or low heteroscedasticity.

Complicating the matter further, some assumptions can interact, which also impacts method validity. For instance, it has been shown, that the violation of the *normality* assumption while performing a *t-test of the Pearson product-moment correlation coefficient* can be tolerated when the variables are *independent*. On the other hand, if the variables are dependent, a violation of normality assumption has severe implications on the t-test's validity [Edgell and Noon 1984]. Therefore, even though some statistical tests are somewhat robust towards violations of assumptions in some circumstances, they still need to be checked to ensure validity.

2.2 Checking for Statistical Errors

For the most part, the best available approach to checking the quality of statistics is expert knowledge. While some journals or fields have agreed upon standards for reporting (*e.g.*, [Kahn 2011]), these do not necessarily address all of the issues raised above. Furthermore, they often depend on authors to self-report that the standards have been met.

There are a few automated tools to address the challenge of missing reporting and inappropriate data analysis. Statsplorer [Wacharamanotham et al. 2015] assists data analysts with automatic testing of necessary assumptions and presents visualizations that aid the user in method selection and interpretation of the results. While Statsplorer supports some methods to conduct statistics, it is not meant to assess the use of statistics in research papers. Another tool, Statcheck [Nuijten et al. 2015], can extract information about null-hypothesis significance testing (NHST) such as p-values, test statistics, and the degrees of freedom reported in publications. The package uses this information to evaluate whether these parameters are in agreement with the claimed results (e.g., whether the test statistic corresponds to the claimed P-value). Statcheck does currently not focus on method-assumption reporting. StatHand [Allen et al. 2016] is a cross-platform application to support psychology students in statistical decision making by virtue of asking a sequence of questions to help them identify a statistical test or procedure appropriate for their goals and data. There are also a plenty of other commercial and open source data mining tools such as RapidMiner, Orange, or Weka dedicated to assist experts in data analysis. However, these software tools are focused on Machine Learning and do not assist practitioners in securing statistical validity of the applied methods.

3 Automating Assessments of Reporting

This section presents an approach that verifies, whether a scientific paper reports testing statistical assumptions for its employed statistical methods. It is implemented as a web-based system.

Verifying, whether statistical assumptions are reported for every applied method goes beyond simple text matching and requires text-understanding. Despite advances in AI and NLP, the problem of computational text understanding on a human-level is yet to be solved. Therefore, in our approach, crowd workers fill this central position. In order to scale despite (expensive) humans in the loop, the costs of using our system must be low. For this reason, our approach is based on crowd workers without special training, which allows us to hire from a very large, diverse pool of workers. The main difficulty therefore was finding a way to let people without knowledge in statistics complement our algorithm.

3.1 Overview

This subsection will briefly introduce the 3 steps of our method, followed by an in-depth explanation of each step.

Given a paper's text, (A) we automatically extract the occurring statistical methods along with their position in the text, the assumptions reported in the paper, and the statistical assumptions that should be tested for. We use a bag of words approach for the first two and a lookup table for the last. Each combination of matches of a method and an assumption in the text is considered a candidate method-assumption pair (denoted by ma). (B) Then, snippets are created for each candidate ma in the paper, followed by asking crowd workers whether the assumption in a ma has been checked for in the text. The snippet contains the part of the paper between the occurring method and the assumption of the ma (see Figure 2 for an example). (C) Lastly, the actual ma pairs are compared with what would have been expected for a given method, according to our rule-base. The results are reflected in the paper.

> To evaluate **H1**, we tested in Study 1 (S1) if there were differences regarding the number of unknown words among CI^2 translations and its 3 base languages. We carried out a one-way (continuous factor) analysis of variance (ANOVA), since we verified that both normality and homocedasticity did hold between groups. Interaction effects were considered at the $p < .05$ level for each of the tested conditions. On the other hand, **H2** was evaluated on the basis of the following criteria:

Figure 2: Example of a snippet for a method-assumption pair ma.
Methods are highlighted in yellow, assumptions are highlighted in green.
In this example, a test of the assumption of *normality* has been reported for the method *ANOVA*.

3.2 (A) Automated Method-Assumption (ma) Extraction

In our first step, we extract all statistical methods and assumptions occurring in the text. To that end, we assume a dictionary[9] δ, which has a list of synonyms for each statistical method and statistical assumption. We then match methods and assumptions in a paper's PDF using Apache PDFBox[https://pdfbox.apache.org/] and regular expressions that account for possible white spaces, page-breaks and hyphenation within each synonym of a method and assumption present in δ. We also assume a rule base σ, which contains, for each supported statistical method m_σ, a list of the statistical

[9] The code and the data are available under an open source license on GitHub at (*suppressed*)

	Normality	Homo-scedasticity	Linearity	No Multi-collinearity
t-test	x	x		
ANOVA	x	x		
MANOVA	x	x*	x	x
ANCOVA	x	x		
Linear regression	x	x	x	

Table 2: Methods and assumptions we used for our validation. Mapping according to [Field 2013]

assumptions $a_\sigma \in A_{\sigma(m)}$ that should be met. Table 2 shows the methods and assumptions used in our work.

Our implementation allows this rule-base to be changed and extended through the administration environment of the web-based system (see Figure 3). Next, for each method m in the paper, for each expected assumption a_σ (according to our rule base $\sigma(m)$), we look for all occurrences of that assumption (i.e., any of its synonyms $a' \in \delta(a_\sigma)$) in the paper's text. For example, if the statistical method assumes normally distributed data, the set of synonyms depicting normal distribution such as *normality* or *bell curve* will be retrieved and matched in the paper.

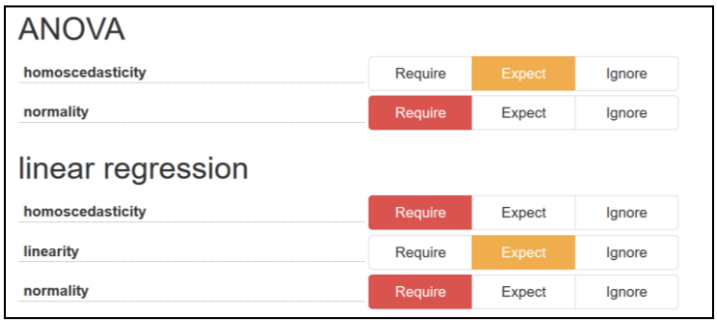

Figure 3: Users can activate / deactivate the underlying assumptions for a given method

All assumptions matched in the text for all methods are considered tuples of candidate method-assumption pairs ma. Consider the following example of an excerpt reporting assumptions in a (mock-)publication: "*Normally, ANOVA is used to assess the difference between two groups. Since our data is not homoscedastic, we used a t-test instead*". The outcome of step A for this excerpt would be 4 ma pairs with the rule base shown in Table 2: (i) ANOVA-normality, (ii) ANOVA-homoscedasticity, (iii) t-test-normality, (iv) t-test-homoscedasticity. Using this rule base, all four resulting ma's should be pruned: (i) and (iii) because normality is a false match of an assumption, (ii) and (iv) because they are reported to not apply. This pruning is an essential step of our workflow and takes place in our next step.

3.3 (B) Human Judgment of Relevance of *ma*-pairs

Once candidate assumptions have been extracted, they need to be checked for relevance to the method m that they are associated with. More specifically, each method-assumption pair ma needs to be checked, such that:

- The assumption candidate a is indeed a statistical assumption (e.g. "normal" is a commonly used word in English)
- The assumption candidate a is semantically relevant to the statistical method m under inspection.
- The assumption candidate a has been reportedly tested for the given statistical method m (to filter cases like "*even though the data was not normally distributed, we executed a t-test*")

Given a tuple ma, the system first generates a snippet containing the portion of the PDF where both relevant terms (method m and assumption a) of an ma-pair are highlighted in different colors (see Figure 2 for an example). We cropped snippets to only show about[10] 3 additional lines above and below the highlighted method and assumption due to copyright regulations. If the relevant method and assumption are more than one page apart, we exclude all intermediate pages from the snippet and offer crowd workers the ability to quickly scroll to either the highlighted method or the highlighted assumption.

Crowd workers were shown a brief explanation of the task, stating that statistical methods have 'prerequisites' (assumptions), where the color used to highlight methods (yellow) and their prerequisites (green) was illustrated. Below was the snippet, followed by two questions. First, we asked '*in the text above, is there any relationship between the prerequisite and the method*'? This question was again accompanied by a hint, giving examples of direct relationships (such as '…we tested [PREREQUISITE] before we used [METHOD]…') and an indirect relationship (such as '…our data were tested for [PREREQUISITE]. Using [METHOD]…'). If the worker answered with "*Yes*," then we asked '*Did the authors of the text confirm that they have checked the prerequisite before applying the method*.' Finally, we asked why participants selected the answers they chose, and whether they felt uncertain. Additionally, we asked crowd workers to rate their confidence in their own answer on a Likert scale from 1-7.

For increased reliability, we employed a variable number of crowd workers for every ma depending on crowd-disagreement and aggregated their results. More specifically, we use the crowdsourcing library PPLib with Beat-By-K and K=3 to determine the number of crowd workers required based on crowd disagreement during runtime (see [de Boer and Bernstein 2016b] for more details on PPLib or Beat-By-K). Beat-By-K can be seen as an extension to a Majority Vote, which continues asking more crowd workers to submit votes until the most popular item has at least K more votes than the 2 most popular item. Crowd workers could not submit more than one answer per snippet. In addition, we only used answers with confidence above or equal 5. This threshold was identified in preliminary experiments, a higher threshold would likely increase the accuracy of our approach, but lead to higher expenses.

Because assumption testing is often reported close to where statistical tests are mentioned, our implementation sorts all candidate assumptions for a given method by distance[11] and sequentially asks crowd workers for their judgments. As soon as a snippet is encountered where the aggregate answer (over many crowd workers to ensure answer quality) to both questions is YES, the corresponding assumption is returned as outcome. The remaining ma candidates for this method with the same assumption (matched at a position farther away in the text) will not need to be tested.

3.4 (C) Reflecting Results Back

When the process is complete, the system shows results for all statistical methods of the paper. Statistical tests are shown in *green*, if all defined assumptions have been mentioned, *yellow*, if some assumptions marked as expected have not been mentioned for a given method, *red* if some assumptions

[10] depending on font-size. 0.6 inches on the paper

[11] i.e., the number of characters between m and a

marked as required have not been mentioned. In each case, the specific statistical assumptions not reported are shown. We also provide a summary screen containing a brief overview over the results as well as other related information (see Figure 4).

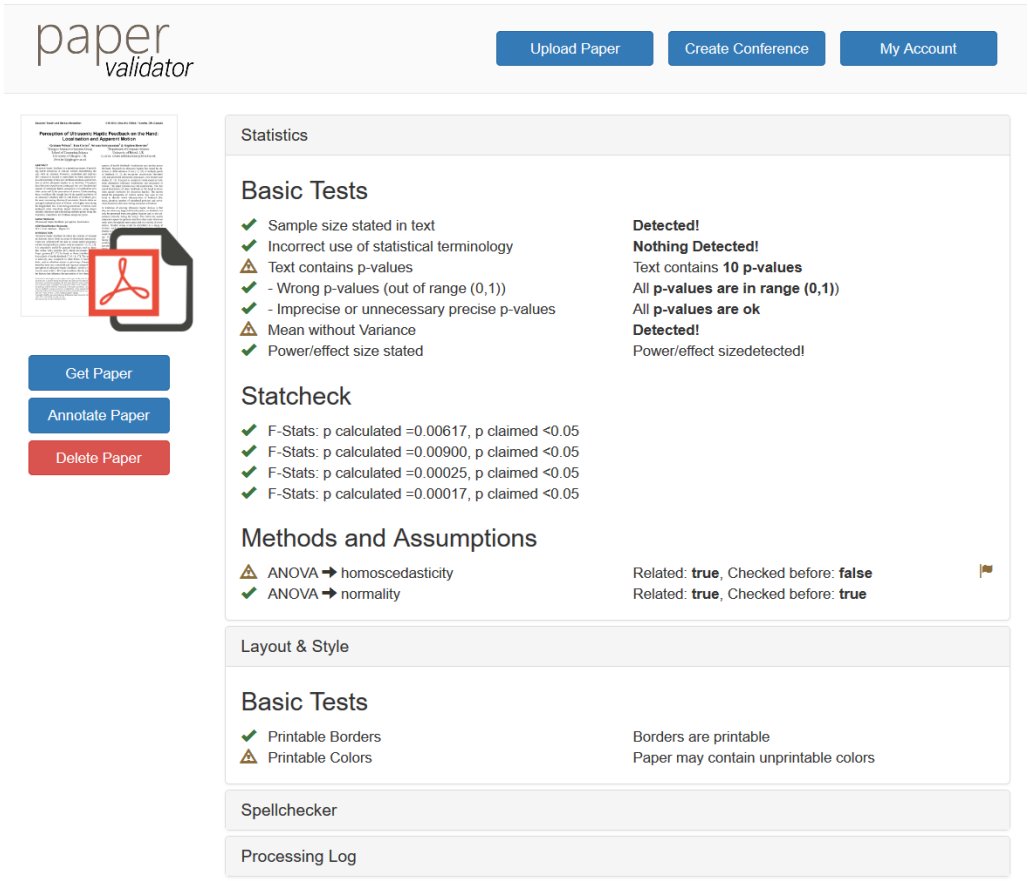

Figure 4: Summary view of a paper on web platform. ANOVA has been used in the paper, and its normality assumption has been reported. The homoscedasticity assumption hasn't been checked

4 Validation

We validate our system on real-world papers published in CHI's proceedings by comparing its output to a manually established baselines

4.1 Overview

We obtained the papers published in CHI (full papers and notes, no posters, keynotes, *etc.*) between 1989-2016 (5132 in total). The use of statistics in CHI follows a long-tail distribution, where few papers use many statistical terms[12] and the vast majority use very few of them (see Figure 5).

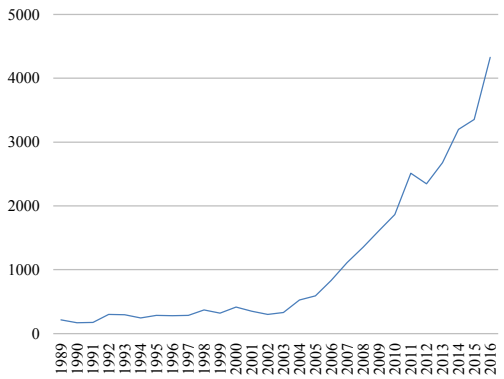

Figure 5: Statistical terms[12] used in CHI

To find the most prominent statistical tests, we analyzed the use of 38 common statistical methods and calculated the fraction of papers per year using each method. Out of these 38 common methods, Figure 6 shows the methods occurring at least in 1% of the papers over the years in median. *ANOVA* is the most common test (median 27% of papers use it), followed by *t-test* (8%) and *Linear regression* (3%). Figure 7 shows the fraction of papers using these methods over time. To limit cost, our analysis is focused on the 3 methods used most over the years: *ANOVA* (and its derivatives *MANOVA* and *ANCOVA*), *t-test* and *linear regression*.

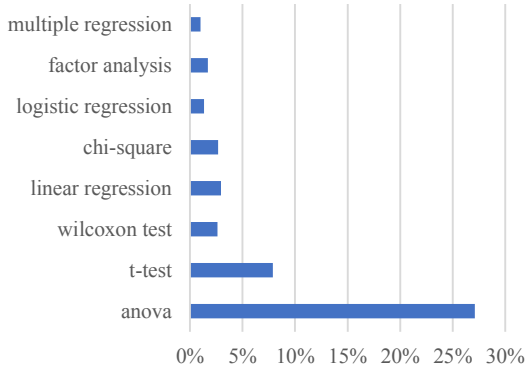

Figure 6: Share of papers using a given statistical method in CHI between 1989-2016. Only includes methods used a median of at least 1% over the years

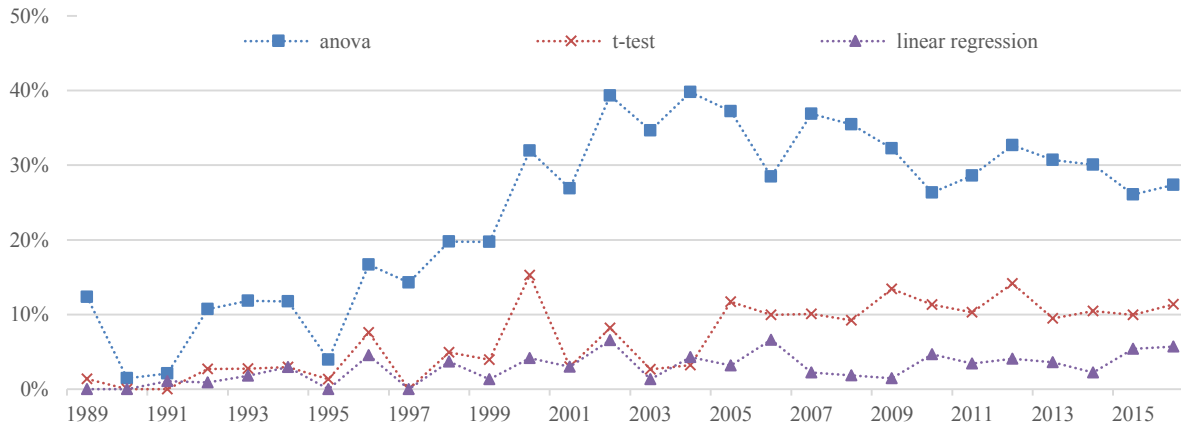

Figure 7: The percentage of papers per year that mentioned each of the top three methods. A clear increase can be seen over the first two decades of the conference

[12] As statistical terms, we used the set of article names part of Wikipedia's 'statistics' category.

Additionally, our approach is compared to 2 other approaches: *Auto_text* automatically accepts an assumption as reported for all statistical methods requiring it, if it occurs somewhere in the text. *Auto_page* automatically accepts an assumption as reported if it is mentioned on the same page as the statistical method in the text.

4.2 Sampling & Ground Truth

We narrowed our sample to the 1671 papers that used the methods we had chosen. We sorted papers by number of mentions of methods, and selected the 50 papers using the most statistical terms and the 50 papers using the least statistical terms for our sample (*validation* dataset). It was our expectation, that increased use of statistical terms implied increased expertise. Thus, sampling from both ends of the spectrum should increase the likelihood that we had included both good and bad examples in our sample, which would help us to better test our tool.

In order to assure the statistical power of our analysis (90%) we calculated the required sample size using Equation 1. This formula is used when the estimated parameter is a proportion. More specifically, in our case, the accuracy of our approach.

$$n \geq \left(\frac{z}{m}\right)^2 \cdot \hat{p}(1-\hat{p}) = 191 \geq \left(\frac{1.64}{0.05}\right)^2 \cdot 0.77(1-0.77) \qquad \text{(Equation 1, [Wild 2000])}$$

Here, z is the cumulative normal distribution at 90% confidence (=1.64), m our targeted error margin of 5% and $\hat{p}$ is the estimated accuracy of our approach. Minimum accuracy calculated across all of our pilot experiments was 77%, which we conservatively used as $\hat{p}$. The resulting n is therefore 191 ma pairs. In preliminary experiments, we observed that papers had an average of two candidate ma pairs. Therefore, we had to sample a total of $191/2 = 96$ papers to meet our power-requirements. To ensure statistical power, we rounded the result up to 100 papers and sampled equally across the two categories *HIGH* and *LOW*.

We manually annotated the selected papers with ground truth labels. Three of the authors with advanced statistics training annotated the papers prior to running the analysis. Each labeler was given a partly overlapping subsets of the sample, so we could assess the inter-rater agreement to ensure the objectivity of the task. The annotators were given a PDF for each paper and a list with methods and corresponding assumptions (see Table 2). Then, they were asked to write down all reported (i.e., valid) Method-Assumption pairs in a paper, as well as highlighting the individual occurrences both parts of the pair in the PDF. Since few papers reported on their assumptions, all papers reporting at least one assumption were later given to all annotators. For papers where at least one assumption was reported, we calculated the inter-rater agreement using Fleiss' Kappa, which was 0.79. This is conventionally interpreted as *substantial agreement*. As a baseline for our system, we then used the result of a majority vote among the three candidate annotations for each ma pair.

We executed our system on the sample and compared its results with our ground truth using standard techniques from Information Retrieval by casting our evaluation to a binary classification problem.

4.3 Data Collection

We used MTurk, hiring only US workers with less than 4% rejected HITs and more than 4000 approved HITs. All experiments were randomized and conducted on mornings of working days Eastern Time. Each MTurk task was priced at 40 cents and for each ma pair a crowd worker could only give one answer.

Our system produced ratings for all ma pairs found in the paper, i.e. a rating for all combinations of methods and assumptions found in the paper, whereas ratings were boolean: "*reported*" and "*not reported*". Our labels provided ground truth for each of these pairs. We calculated *Precision* (share of

correctly classified *ma* pairs out of all returned *ma* pairs), *Recall* (share of returned *ma* pairs out of all relevant *ma* pairs), *Accuracy* (correctly classified *ma* pairs out of all relevant *ma* pairs), and *Fleiss' Kappa*.

When annotating the 100 papers with ground truth, we identified 251 candidate method-assumption pairs, whereas 82 of them were rated as "*reported*". As expected, the distribution of encountered method-assumption pairs is highly skewed: because 89 out of the 100 papers do not report any assumptions at all. Even in the 11 papers that reported checking some assumptions, only 50% of them were checked on average

The total cost of running the system on these 50+50 papers was $214 USD.

Upon completion, we excluded 2 papers out of these 100 from the analysis, both of which had an extraordinarily high error rate in our tool. Both papers were discussing statistical methods at a meta-level (*e.g.,* presenting a tool for helping people do statistics) rather than applying these methods. Since our system did not ensure that occurring statistical methods are indeed used by the authors for data analysis, such papers had a high error rate. Our system could easily be extended for this case, by posing a corresponding question to the crowd workers of each snippet.

4.4 Results

Compared to the ground truth, our system achieved a precision of 0.82 with a recall of 0.83. This means, that 82% of the identified method-assumption pairs were also present in the ground truth, and that 83% of the *ma*-pairs in the ground truth were identified by the system. We therefore achieve an F1-score of 0.83. We also calculated Cohen's Kappa, which was 0.70. Table 3 shows the confusion matrix of our validation.

		Predicted	
		ma reported	*ma* not reported
Actual	*ma* reported	68 (TP)	14 (FN)
	not reported	15 (FP)	101 (TN)

Table 3: Confusion matrix of system when compared to ground truth. 68 *ma*'s were correctly labeled as reported in their papers (TP = true positive), while 15 *ma*'s were mistakenly labelled as reported by the system but actually haven't been in their papers (FP = false positive).

Both baselines perform worse than our system. *Auto_text*, our generous baseline, easily accepting assumptions as reported, achieves a precision of 0.43 and a recall of 1.0. Note, that a recall of 1.0 is not good on its own, if the precision is low. This amounts to an F1-score of 0.63. Kappa could not be calculated for *Auto_text*, since the Precision was too close to chance. *Auto_page*, the less generous baseline using heuristics to accept assumptions as reported, achieves a slightly higher precision of 0.66 with a recall of 0.74. Cohen's Kappa for *Auto_page* was 0.40.

The reason in both cases is the baseline's strategy for the filtering of *ma*'s, for which our approach uses crowdsourcing. *Auto_text* does not filter *ma*'s at all, and *Auto_page*'s approach only accepts assumptions as tested if the occur close to the method in the text. This could lead to wrong results, for example, when papers stated that they have tested statistical assumptions for all of their methods in the experimental setup section as opposed to right by the method.

4.5 Failure analysis

To better understand the limitations of our approach, we asked crowd workers to include a (free-text) justification with each of their submissions. Since there were more than 500 answers, we read through an excerpt of the *ma*-submissions with wrong answers (as qualified by our annotated baseline) and synthesized three basic sources for errors: (i) crowd workers who submitted their answers very quickly and therefore lacked time to give it enough thought, (ii) crowd workers who did not understand what

we meant by *method* and *assumption*, and (iii) crowd workers who misunderstood the task description. For example, some workers thought about sentence structures instead of the semantic relationship between the two marked terms in a snippet.

5 Statistical Assumption Reporting in CHI

The rarity of statistical assumption reporting in the sample we used to validate our method was striking. However, we were curious whether that trend held in a larger sample of CHI papers and how it changed over time. Specifically, we wanted to shed light on the following questions:

RQ1.1: Use Indicates Expertise: Does increased use of statistics coincide with increased expertise? This can be tested by correlating the number of method-assumption pairs reported against the number of methods (out of the list of 38 common methods) mentioned in a paper. In other words, we wanted to check, if authors using a diverse set of statistical methods in their papers tend to be more attentive to assumption reporting.

RQ1.2: Quality Improves with Time: Did the quality of statistical reporting (as represented by the reporting of assumptions) improve over time? This can be tested by a linear regression with the number of assumptions reported in papers as a function of years. Another, more discretized, evaluation would be comparing the number of assumptions reported in papers published in later decades of CHI to those published in earlier decades.

In order to better quantify this effect in CHI, we analyzed papers published between 1989 and 2016. For cost effectiveness, we focused our analysis on the two most popular methods (ANOVA and t-test) and sampled 30 papers randomly drawn each year from those using either the ANOVA or t-test in clusters at five year intervals starting at 1989 (*i.e.,* the years 1989, 1994, 1999, 2004, 2009, 2014). Due to the smaller size of CHI proceedings in earlier years, 1989, 1994 and 1999, only had 8, 9 and 17 papers using these methods. We therefore increased our sample to additionally include 1995, 1996, 2005, 2006, 2015 and 2016 totaling to 261 papers across all years under analysis (1995 and 1996 had only 3 and 14 papers respectively).

5.1 Method

To answer our research questions, we used our system to obtain the *reported* as well as *expected* assumptions of the methods applied in sampled papers. The total cost was $287 USD. Note that the results presented below have to be seen in the light of the Precision and Recall (82% and 83% respectively) of our system – implying that the results can err to some extent. However, even human annotators could be expected to make some errors as evidenced by the inter-rater agreement (Fleiss' Kappa) of .79 achieved by the authors when establishing a baseline (Cohen's Kappa of our system is .7). Thus our system's performance is close to the best available alternative, human reviewers typical of CHI.

To answer RQ1.1, whether **Use Implies Expertise,** we calculated the number of assumptions mentioned in each paper (for ANOVA and t-test, if mentioned) and the number of statistical tests mentioned in each paper (including all 38 methods from our original analysis). These were compared using the Spearman rank correlation (ρ_s, which does not require the data to follow the normal distribution). Note, that by using the number of reported assumptions in a paper instead of the fraction of reported assumptions, we reduce the dependency on our rule base σ.

To answer RQ1.2, whether **Quality Improves with Time,** we look at two analyses: (A) did individual papers show an improvement as to the share of assumptions reported, (B) is there an improvement of the fraction of papers per year reporting at least one assumption.

For (A), in each paper, we calculated the number of assumptions mentioned divided by the number of assumptions that would be expected (the "*checking rate*"). The number of expected assumptions for each method is obtained through our rule base σ (see Table 2). We then fitted our data with a linear regression. A positive slope would mean an improvement over time (if the model is significant). Our data for this regression violates its normality assumption and its homoscedasticity assumption, which strongly reduces our confidence in our model. We therefore ran an additional test where we first dichotomized the data, thus reducing the resolution. More specifically, we grouped the whole sample of 261 papers into two roughly equally sized groups: 141 CHI papers before 2009 and 120 CHI papers in or after 2009. The year 2009 was chosen because it divided the sample in the most balanced groups. We then used a Mann-Whitney U test to check if the latter group showed an increase in reporting. Mann-Whitney U is non-parametric and therefore does not assume normality. While originally proposed without requiring homoscedasticity, many statisticians currently argue about lower confidence in results in cases of unequal variance [Vargha and Delaney 1998]. Zimmerman [Zimmerman 2004] quantifies this effect for various proportions between the standard deviations of the two groups. In our case, this proportion is 1.79. For a proportion of 2, Zimmerman finds a 8.3% probability of a Type 1 error (falsely rejecting H_0) at N=25 and α=0.01. To quantify the effect size, the Rank Biserial Correlation is often used for Mann-Whitney U.

Another part of the same research question is, whether the aggregates have changed over time. (B) we therefore created a linear regression on the fraction of papers reporting at least one assumption for each year. Note, that step B decouples the analysis from our rule base σ and is therefore more generous than step A. However, it may also be skewed, due to smaller sample size in the earlier years. We therefore tried mitigate this issue by following it up with a weighted linear regression, where we weight the fractions of each year by the number of samples in that year. We visually inspected the dependent variable used in this regression (QQ-plot and residuals vs fits values plot). Even though the plots do not conclusively prove the assumptions to be met [Osborne and Waters 2002], violations are fairly mild and may result out of the small sample size (N = 12 years). The reported results should therefore be seen in the light of these constraints.

5.2 Results

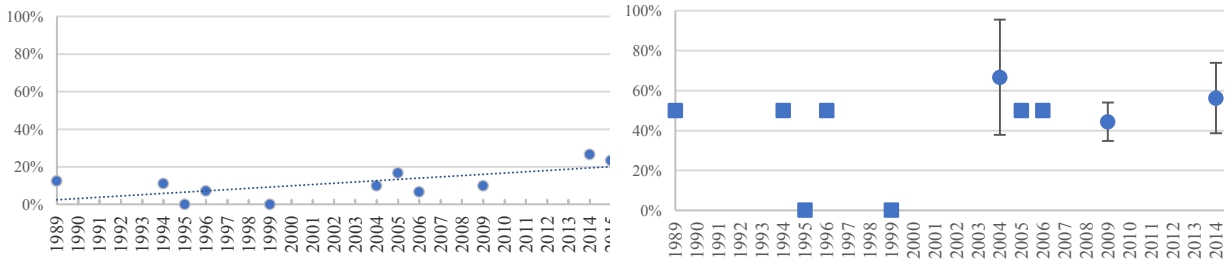

Figure 8: Analysis of the reporting of assumptions in CHI: (Left) Share of papers reporting at least one assumption in CHI over the years. (**Right**) Mean percentage of papers that check at least *one* assumption, (100% would mean that in that year, all assumptions were mentioned). The error bars (mean ± 1 standard deviation) are only shown for years with more than 3 papers reporting at least one assumption (circles).

Figure 8 shows the share of papers checking at least one of their assumptions over time on the left. Only 35 of the 261 sampled papers report at least one of their assumptions. A slight upward trend towards more papers reporting some of their assumptions is visible on the left graph of Figure 8. Among the papers reporting assumptions, roughly half of the expected assumptions are tested as shown on the right graph (Figure 8 on right).

Use Implies Expertise: We found no correlation between the number of statistical methods used in a text and the number of assumptions reported (ρ_s=0.136, p=0.028).

Once the papers that do not report any assumption at all are excluded, a small correlation is revealed (ρ_s=0.516, p=0.015). Figure 9 shows the scatter plot for this case.

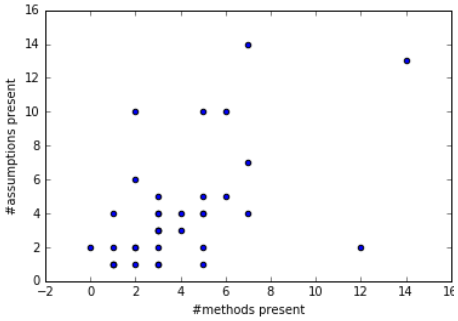

Figure 9: Scatterplot of the number of assumptions present in papers in relation to the number of methods

Quality Improves with Time: The regression on the paper's *checking rates* shows a small improvement over time, but only 4% of the variance can be explained with our linear model (R^2=0.04, p=0.001, slope=0.01). Recall that this analysis does not meet its statistical assumptions, which further reduces explanatory power. Hence, we also ran the Mann-Whitney-U test. This test shows a slight difference between assumption reporting prior to 2009 and afterwards (p=0.001). A Rank Biserial Correlation quantifies the effect size as very weak ($r_{rb} = 0.14$).

A linear regression on the *fraction* of papers *reporting at least one assumption* over time (see Figure 8, left) seems to point to a small increase in the share of papers reporting assumptions over time (R^2=0.68, p=0.01, slope=0.0066=0.66% increase per year). When weighting each year by the sample size in that year, the effect is slightly increased, at a lower R^2 (R^2=0.58, p=0.004, slope=0.0089=0. 89% increase per year).

Discussion: Since, only a small fraction (13%) of papers report checking assumptions, it is difficult to draw strong conclusions from these data (except that CHI may wish to reconsider its standard for what is expected with respect to statistical – and particularly statistical assumptions – reporting).

Whether or not the community is already making a change as issues with NHST become more visible at large is unclear. Reflecting on the quantitative results of our study with necessary caution, it would be wrong to reach any definitive conclusion. However, our data seems to indicate a weak trend developing towards a larger fraction of papers reporting at least one assumption.

6 Threats to Validity

The threats to validity can be classified into 2 categories: sample-related and approach-related. While the former is concerned with the statistical power of our analysis, the latter can have negative impact on the reliability.

Sample-related: Our sample of at most 30 CHI papers per year may not be enough to draw a conclusion. This issue is aggravated in earlier years of CHI, where fewer papers used the statistical methods in scrutiny. This problem could be addressed in future work by including more statistical methods and hereby broaden the scope of the analysis.

Second, in similar vein, our analysis of CHI papers over the years is limited to the two most commonly occurring tests with assumptions – the t-test and ANOVA. It is possible, that CHI papers are better at reporting assumptions for other, less common statistical methods. However, our focus on the two most prominent methods ensures that our insights apply to the majority of CHI papers using NHST.

Third, in hindsight, the assumption made for the power analysis of the *validation*-dataset, that every paper would have about two assumptions, was somewhat oversimplified. Ultimately, our power goal was still reached though.

Lastly, our system did not find a paper checking at least one of its assumptions in 1995 and 1999. We therefore manually checked each paper in our sample of these years, to ensure that the results were correct.

Approach-related: While the validation of our system covered both, step A (extraction of *ma*-pairs) and step B (filtering unreported *ma*-pairs), it is possible that the analysis of CHI over time would have required different sets of synonyms (*e.g.,* assumptions might have been addressed differently in earlier years unbeknownst to us). Or, more generally, that our rules for identifying assumption reporting may be too conservative (relevant for step A).

Second, the current version of our system does not differentiate between the mention of a statistical method as a topic of investigation (as in the two papers, which we had to exclude in our *validation*-dataset) and the mention of a method for use for analysis. Whilst we corrected for this error when testing the validity of our approach, we did not do so in our CHI analysis, which may add another source error. We do believe, however, that such an effect would be limited and equally distributed over the years. As mentioned, this could be fixed in a next version of our system.

Third, our crowd process may be unreliable, returning different results when being given the same inputs. While non-determinism in a crowd setting is well-known, a strong variance of the process would reduce its generalizability. However, our validation indicates that the variance is close to the variance of human raters. We would need to repeat this testing for other domains to ensure the generalizability of our method.

7 Conclusions and Future Work

In this paper, we proposed a system to automatically determine the reported statistical assumptions in research papers. We used this system to infer the reporting of assumptions at CHI between 1989-2016. We found that very few papers test the underlying assumptions of their statistical methods. The vast majority (~87%) even report no assumptions whatsoever. Our data indicates a slight improvement towards more papers reporting at least one of their assumptions over time. The low improvement rate over time clearly indicates that a discussion of reporting standards for both statistical methods and their assumptions might be warranted within the CHI community. An inclusion of a system such as ours into the reviewing process may help to ensure that such standards would be upheld and may pave the way for better assumption reporting, hopefully leading to higher statistical validity at CHI.

In future work, we plan to extend the functionality of our system. The primary goal would be to provide a more detailed analysis for a more diverse set of reporting issues. Second, we intend to ascertain the system's generalizability beyond the CHI domain by testing others. Preliminary results show that in medicine, more statistical assumptions are reported per method than in CHI, which may be explained by the traditional presence of (bio)-statisticians in medical research.

We see our paper in line with other recent publications discussing statistical standards at CHI. Specifically, we draw attention to the need to define a clear expectation as to the testing of the underlying assumptions of frequently used methods. Whether the reader believes, that testing underlying assumptions is crucial or takes a stand in favor of less restrictive statistical testing, a public reflection and discussion on this point is necessary. Our openly available system could be employed to increase the adherence to the rules that results from this discussion. If deployed, the systems could be used (i) as a means to reduce the load on expert reviewers by prescreening and annotating submissions, (ii) by paper authors as a convenient check whether they meet a conference's assumption reporting standards, and/or (iii) by conference chairs to get an overview over assumption reporting across all submissions. Note that we propose to use the system in conjunction with human experts to address its misclassifications. Lastly, while NHST has been criticized, our system may help mitigate some of its pitfalls until our community decides to switch to a more sustainable method for validating claims. Whatever the use, we hope that, if sparked, the ensuing discussion will allow us to stand on the shoulder of giants with confidence.

8 Acknowledgements

We would like to thank Mattia Amato for assisting us with preliminary analysis paving the way for this paper. This work was supported in part by the Swiss National Science Foundation (SNSF- Project: 200021- 143411/1).

PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation

This chapter is based on a paper that has been published at the journal **ACM Transactions on Intelligent Systems and Technology** [de Boer and Bernstein 2016b]. We presented a short version of it at the Collective Intelligence conference 2015 as a poster [de Boer and Bernstein 2015].

PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation

Abstract

Crowdsourcing is increasingly adopted to solve simple tasks such as image labeling and object tagging, as well as more complex tasks, where crowd workers collaborate in processes with interdependent steps. Research has yielded numerous patterns for coordinating crowd workers in order to optimize accuracy, efficiency, and cost. Process designers, however, often don't know which pattern to apply to a problem at hand when designing new applications for crowdsourcing.

In this paper, we propose to solve this problem by systematically exploring the design-space of complex crowd-sourced tasks via automated Recombination and Auto-Experimentation for an issue at hand. Specifically, we propose an approach to find a fitting process for a given problem by defining a problem in terms of its abstract operators, generating all possible alternatives via the (re-)combination of the abstract deep structure with concrete implementations from a Process Repository, and then establishing the best alternative via Auto-Experimentation.

To evaluate our approach, we implemented PPLib (pronounced "People Lib"), a programming framework that allows for the automated recombination of known processes stored in an easily extensible Process Repository. We evaluated our work by generating in two real-world examples on Amazon's Mechanical Turk. We close the paper by comparing the two settings where the Recombinator was used, and empirically show that crowd processes performance (of the same processes) varied widely among the two. We therefore contend that there is no unifying formula for crowd process design, which emphasizes the need for an automated solution like PPLib.

1 Introduction

Following the industrial revolution, labor was increasingly structured into business processes, where a person or a group of people specialize on a part of the overall product. In crowdsourcing, a similar development can be observed as crowd workers participate in structured crowd processes that aim to get work done as accurately, efficiently and inexpensively as possible. To enable this, a number of design patterns (or *crowdsourcing patterns*) have been established, including Find-Fix-Verify [Bernstein et al. 2010a], majority voting, and Iterative Refinement [Little et al. 2010].

However, there are currently no reliable guidelines on when to use which pattern, or even on how many crowd workers should be employed at each step in a crowd process. Essentially, crowd process designers are left to pick and implement one of the patterns *purely on intuition,* without any assurance that it is a suitable choice for the problem they are facing.

In the realm of Business Process Reengineering (BPR), [Bernstein et al. 1999] introduced the *Process Recombinator* through which novel business processes could be generated from a repository of existing processes. As a first step, their approach calls for process designers to identify the process deep structure – i.e., the main abstract processing steps that make up the core activities of the process. They then rely on the paradigm of specialization from object-oriented programming and F-logic [Angele et al. 2009] to enrich these descriptions into concrete instantiations of the desired processes. This approach regards the deep structure of the process as a specification of the design space [Ulrich and Eppinger 1995] and the recombination into its instantiations as a systematic exploration of this design space. Unfortunately, given the Process Recombinator's application domain in BPR, it did not generate

executable processes, as this would involve reorganizing corporations – a sizable effort that is hardly practical.

In realm of crowdsourcing, it is much easier to try different candidate processes for a problem at hand. Hence, we extend the idea of process recombination with a dependency injector [Fowler 2004] that generates executable crowd programs that can then be automatically tested and evaluated in a crowdsourcing market such as Amazon Mechanical Turk[13]. Specifically, we propose to not only systematically generate the whole design space of crowd programs as specified by a process deep structure, but also to automatically run experiments to determine which alternative is the best[14] choice – thereby "reducing" the problem of designing complex crowdsourcing solutions to that of defining their deep structure and process performance metrics. The Recombination and Auto-Experimentation process can then decide, which instantiation promises to be the most successful.

As a consequence, the main contributions of this paper are as follows:

- We propose a novel methodology for systematically exploring the design space of crowd-computing processes that combines process recombination with Auto-Experimentation.
- We introduce a system called PPLib (a recursive acronym that stands for PPLib Programming Library) that incorporates this methodology. By leveraging the PPLib Process Repository (PPR)—a systematically organized repository of crowd-process patterns and process fragments—PPLib is capable of generating all combinations of the different process implementations to exhaustively search and evaluate the space of candidate processes in order to **find the best[14] process for a given problem**.
- We introduce a first version of our repository of existing abstract crowd processes that can be used as building blocks for recombination.
- We release PPLib and PPR under an Open Source License on GitHub[15], where we also provide documentation and show how applications can (re-)use our approach, our software and our process repository.

We evaluate our methodology and its associated library by showing that PPLib is capable of independently designing and finding non-trivial, well-performing, robust crowd processes for well-known crowd-computing problems with objective solutions (i.e., tasks with correct or incorrect outcomes) such as text translation and text shortening.

2 Related Work

In the following we point to related work in the different fields our research touches upon: Business Process Reengineering (BPR), Crowdsourcing and Auto-Experimentation.

[13] http://www.mturk.com

[14] The measure of suitability can be defined by the user through a utility function. Typical parameters for such a utility function may include cost, duration, and adequacy of the result.

[15] https://github.com/uzh/PPLib

2.1 Business Process Reengineering

In the late 1990's the topic of business process reengineering (BPR) and management [Hammer and Champy 1993] rose to great prominence. BPR was seen as an approach to modernize organizational processes in corporations and turn them into more customer focused institutions. Extending on ideas of the Tayloristic method and early approaches to high-level process automation [Hammer et al. 1977; Zisman 1978] numerous business process modeling approaches were proposed [Lee et al. 2008]. These modeling approaches still left the question of how to systematically innovate new approaches. [Malone et al. 1999] proposed the use of a Process Handbook, an electronic catalogue of organizational processes, to inspire innovation. The MIT Process handbook [Thomas W. Malone et al. 2003] encompassed 5000 business process that were organized according to principles of coordination theory [Malone and Crowston 1994] as well as a specialization hierarchy, which used a special type of inheritance to organize processes in a taxonomy [Battle et al. 2005]. The advantage of this organization was that non-BPR-specialists were able to browse the handbook for inspiration and add novel processes.

Inspired by product design and management approaches [Ulrich and Eppinger 1995], Bernstein *et al*. [Bernstein et al. 1999] proposed to use the repository of the process handbook as basis to automatically recombine different elements of processes to be able to explore the whole design space of business processes. Specifically, they proposed that innovators define business processes using abstract processes from the process handbook, and then established that a Process Recombinator would be able to generate all processes of this design space. The resulting process candidates could then be used as an inspiration to establish innovative and new processes in companies - shifting the main focus of BPR from process design to choosing among the most useful alternatives. The authors concede that one of the main limitations of their work is that process recombination may lead to too many candidates and that it is difficult to support an automated pruning of the candidates in the realm of BPR.

2.2 Crowdsourcing

Academia has provided many innovative methods to engage with such on-demand workers. [Little et al. 2010] proposed Turkit, a scripting language that allows its programmers to easily interact with crowd workers on Amazon's Mechanical Turk platform. They focused on how the high latency of human computation affects writing and debugging such code, which led them to propose the Crash&Rerun programming pattern, allowing the programmer to repeatedly rerun and debug processes without needing to republish costly previous steps. [Franklin et al. 2011] used a similar approach in CrowdDB to delegate joins to human workers. [Tranquillini et al. 2015] developed CrowdComputer, a programming language for crowdsourcing and equipped it with a visual editor to easily create crowd processes in BPR. Some areas within crowdsourcing have seen the advent of tailored frameworks: Medusa [Ra et al. 2012] poses an example for the area of crowd sensing.

Another group of frameworks focus on the Map-Reduce approach to manage highly interdependent tasks. [Kittur et al. 2011]'s CrowdForge is among the main constituents of this group. They split down large problems into sub-problems, using either algorithms or other crowd workers. Once these sub-problems are solved by machine agents or human agents, the results get collected and aggregated.

[Ahmad et al. 2011] expanded on this idea and proposed Jabberwocky, which added the possibility of using a powerful high-level procedural language to process tasks.

Programming in a computer language to essentially coordinate human workers and CPUs equally led [Bernstein et al. 2012] to think of the Global Brain–metaphor, where they combine networked humans and computers in a heterogeneous graph. Automan [Barowy et al. 2012], an automatic crowd programming system, fits this metaphor well and integrates human computation tasks as function calls

in a standard programming language, hereby blurring the lines between CPUs and human processors. Among the main problems with programming the global brain is the fundamental difference between humans and computers in terms of motivational, error, and cognitive diversity [Bernstein et al. 2012]. Furthermore, many problem-solving processes are difficult to specify ex-ante and only gain more specific definitions during executions. [Bernstein 2000] proposed the notion for tasks to move along *the specificity frontier*, from well-defined and static to loosely defined and dynamic. While Automan requires processes to be very well-defined, which therefore leads to static processes, applications developed using Turkomatic [Kulkarni et al. 2012] are located on the opposite end of the specificity frontier. In Turkomatic, a programmer specifies her high-level goal in plain English and crowd workers are then employed to formalize the description – and execute it.

In order to address the diversity of human error, research has identified quality assurance as an important problem in crowdsourcing. Automan, for example, treats answers of multiple choice queries as samples of a statistical process and continues sampling until a predefined confidence-value is reached. Similarly, Beat-By-K continues sampling until the item receiving the most votes has at least K more votes than the second placed item [Goschin 2014]. [Livshits and Mytkowicz 2014] calculate the right amount of samples to be taken for each query ex-ante using power analysis. [Bernstein et al. 2010a] proposed the popular Find-Fix-Verify pattern, in which workers would first be presented with a list of options where they mark entries that are to be refined. In the Fix step, a number of workers generate alternatives to these entries, out of which the best one is chosen in the verify step. They have deepened the idea of Find Fix Verify in Context Trees [Verroios and Bernstein 2014], where work is divided & conquered on a tree-like structure. In Iterative Dual-Pathway [Liem and Chen 2011], workers are assigned to one of two groups. Both groups are working on the same tasks in parallel but don't see the other group's answers to the same task. Results are then verified by comparing the answers of the two groups for a single task. [Inel et al. 2014] proposed CrowdTruth, a framework that uses the disagreement between crowd workers to evaluate data quality, question ambiguity and worker quality.

We have proposed CrowdLang [Minder and Bernstein 2012], a workflow-based language that enabled *manual* recombination processes. This contribution expands on our earlier idea by presenting an *automated* recombination mechanism that works on a process repository and combines it with Auto-Experimentation for selecting successful crowd programs.

2.3 Auto-Experimentation

Automating science is a long-standing dream of scientists in many disciplines. The closest to full automation of the scientific endeavor is probably the Robot Scientist called "Adam" [King et al. 2009], which autonomously generates hypothesis, plans and executes experiments to test the hypotheses, and draws conclusions to discover novel findings in yeast genetics. While in most other domains of science such a fully automated discovery process is somewhat futuristic, some domains have adopted automation approaches for a variety of steps in the scientific process. In Knowledge Discovery in Databases (KDD) and statistical analyses, for example, people have explored the notion of Intelligent Discovery Assistants [Bernstein et al. 2005; Serban et al. 2013] – systems that advise human data analysts on which approaches to use in experimentation. Furthermore, this domain has a long tradition to explore alternatives automatically via Auto-Experimentation: the automatic planning, execution, and evaluation of experiments (e.g., [Serban 2010a]).

In crowdsourcing, Auto-Experimentation is most often used to evaluate hypotheses of researchers (e.g., [Sheng et al. 2008]). Crowd programs are scheduled automatically, and evaluated either automatically (through the use of yet other crowd programs) or manually (by a panel of experts). We have, however,

not found any approach that proposes to pair the systematic exploration of the design space of a solution with an evaluation of the candidates via Auto-Experimentation.

3 The PPLib Methodology

Composing crowdsourcing solutions is essentially a human problem-solving task. According to [Newell and Simon 1972; Simon 1977] human problem solving can be organized into four phases: (i) intelligence: defining a problem, (ii) design: composing a variety of alternative choices for the problem, (iii) choice: the evaluation of the alternative solutions to select the best alternative, and (iv) implementation: the implementation of the chosen solution. Inspired by [Bernstein et al. 1999], we modeled our PPLib methodology according to Simon's phases.

1. *Intelligence*: *Identifying the process deep structure.* Simon's first phase of human problem solving calls for the identification and definition of a problem. In AI planning, a generic problem definition is typically given via the specification of the initial and desired state as well as the domain model (i.e., the set of possible actions in the problem domain). In our case, we propose to define a crowdsourcing problem via the specification of its deep structure [Chomsky 1965] – its most abstract activities akin to the top level of a grammar in Hierarchical Task Network (HTN) planning [Ghallab et al. 2004]. Specifically, we propose to define the deep structure of the solution by incorporating building blocks chosen from the PPLib Process Repository (PPR) and/or from a programming language. Elements chosen for the ontology can later be used for recombination [Bernstein et al. 1999].
   Note, that the HTN-inspired approach somewhat blurs the boundary between the first two steps of Simon's approach, as identifying the deep structure of a solution specifies the design space for them, since abstract elements chosen from the PPR indicate the degrees of freedom that can be exploited for process recombination and, hence, for the design of solution alternatives.
2. *Design: Defining parameter candidates.* Our choice of problem specification via process deep structure already outlines the structure of the design space. PPR processes can be configured by supplying parameters such as crowd worker count, confidence value to be achieved in repeated single choice questions, or money to be paid to crowd workers for the tasks involved. Hence, the first element of *design* in our methodology encompasses the specification of parameter ranges. The second element requires the definition of a utility function to rank the solutions of the process candidates resulting from the Auto-Experimentation phase.
3. *Choice: Recombination and Auto-Experimentation.* In order to be able to choose between the possible solution candidates, they need to be instantiated and their performance needs to be evaluated. To fulfil this goal, we propose to explore the design space of possible solutions as follows:
   (i) Generate recombinations. In this first step we take the problem definition specified in steps 1 & 2 and generate all possible instantiations employing the deep structure and the PPR like a production grammar to create surface structures.
   (ii) Run Auto-Experimentation. This second step takes all generated surface structures, runs them, and evaluates the results using the utility function defined in step 2. Note, that in the simplest case, the ranking implied by the utility function would indicate one winning process, while an extended evaluation may find multiple solutions with different trade-offs (such as robustness to various conditions, etc.).
4. *Implementation: Deployment of the chosen process*. Implement the chosen process in the production environment.

This abstract definition of our methodology leaves a number of questions open. For that reason, the remainder of this section will further explain the main elements of the methodology, before we cover implementation details for each of these steps in Section 4, where we also show an example of how to use this abstract definition.

3.1 The PPLib Process repository

The PPLib Process Repository (PPR) stores the crowd computing, human interaction, and general process organization patterns and organizes them in a taxonomic structure. Inspired by the MIT Process Handbook [Malone et al. 1999], we chose to organize them into a type hierarchy, where more special processes can fulfill the role of a more generic ones [Wyner and Lee 2002]. In the style of object-oriented programing we call more general entries that are not sufficiently specified to be enacted (or executed) *abstract crowd processes* and enactable entries in the PPR *concrete crowd processes*.

The PPR's structure is inspired by both the Process Handbook [Thomas W Malone et al. 2003; Malone et al. 1999] and the collective intelligence genomes [Malone et al. 2010]. The PPR currently contains the top-level genomes suggested in the WHAT dimension of their ontology (see Figure 10): CREATE and DECIDE. CREATE is used for different ways of getting crowd workers to create collections of items such as texts. DECIDE is used to select one (or more) element from a collection of alternatives.

All processes in PPR have a specified input type and a specified output type, which also get inherited to ensure polymorphism. Since the PPR builds upon inheritance and polymorphism, one can easily build complex processes that consist of other processes in the PPR.

Note, that the PPR does not claim to be a complete library of all crowdsourcing tasks. For example, the Process Handbook, on which the PPR is based, also suggests COMBINE or DIVIDE as top level processes. We see the PPR as a growing repository, whose usefulness increases as contributors add more building blocks.

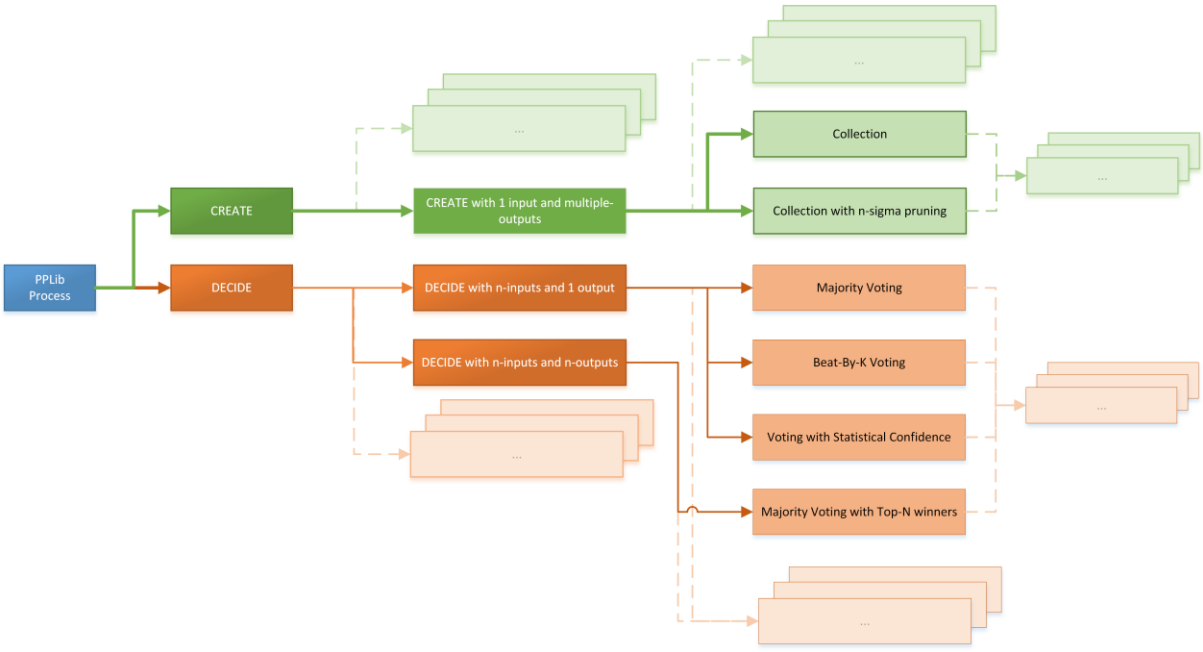

Figure 10: Top Level of the PPLib Process Repository including the basic building blocks

3.2 Process Recombination

Having established the notion of abstract processes in section 3.1, we would now like to show how these abstract processes are used for recombination. We distinguish between two forms of recombination: (A) parameter recombination and (B) workflow recombination.

> *(A) Parameter recombination*: All processes in the Process Repository are configurable using parameters. Often, one would like to try different variants of each individual process, such as different payment values, different questions to ask crowd workers, or varying numbers of worker counts used. For discrete parameter-types, the PPLib Recombinator generates all possible combinations of the specified parameters of an abstract process, where each combination results in its own instance of the abstract process under recombination.
>
> *(B) Workflow recombination*: Workflow recombination leverages the plug-in nature of the specialization hierarchy of PPR to generate alternatives. Consider the case, when multiple Human Computation Tasks are declared in the application under recombination, e.g., $HComp_1$ and $HComp_2$. Here, the Workflow Recombinator looks for all possible alternatives that the PPR offers for both $HComp_1$ and $HComp_2$ and generates them. If, for example, the PPR specifies process P_1 and P_2 for $HComp_1$ and process P_3 and P_4 for $HComp_2$, then the instantiations { $<P_1, P_3>$, $<P_1, P_4>$, $<P_2, P_3>$, $<P_2, P_4>$ } are generated. Note that this process is recursive. Hence, if P_2 contains the steps $P_{2.1}$ and $P_{2.2}$, where $P_{2.1}$ is an abstract process that has two alternatives *P5* and *P6* then the list above is extended to the following resulting processes: { $<P_1, P_3>$, $<P_1, P_4>$, $<<P_5, P_{2.2}>, P_3>$, $<<P_6, P_{2.2}>, P_3>$, $<<P_5, P_{2.2}>, P_4>$, $<<P_6, P_{2.2}>, P_4>$}.

It is evident that generating all combinations of processes with their parameters has exponential runtime and space complexity – this requires a process designer to think carefully about which parameters and/or workflows to recombine. Hence, we also allow process designers to specify restrictions on which recombinations are actually used by specifying parameter ranges or valid specializations explicitly in the deep structure.

3.3 Auto-Experimentation

Once all requested process candidates are generated, they can be evaluated in parallel by executing each candidate on a representative sample of the given crowdsourcing problem and measuring the adequacy of their results. For example, if the overall goal is to translate a book trough crowdsourcing, one could evaluate the fitness of a crowd process candidate by using it to translate a single page from that book.

The Auto-Experimentation Engine executes all recombined processes on a sample, retrieves their results, and applies the user-defined utility function to it in order to rank them. More formally, given a set of recombined crowd-processes P, a specified number of iterations the experiment should be repeated i, and a utility function f: Execute each process $\in$ i times and apply the utility function f to each result obtained in the individual iterations. In the end, rank[16] all by their mean[17] utility as evaluated by f across all i.

[16] Note, crowd process performance is established through ranking. To increase efficiency, we will explore the use of mathematical optimization on page 59

[17] Other aggregation functions (such as median) suitable for ranking could be used here as well.

The sample size for each p is therefore determined by i and should be chosen large enough, such that one can derive scientifically sound findings from the results – which leads to a tradeoff between the cost and accuracy of the experiment.

Please note that PPLib leaves the definition of f entirely to the user. Possible factors to include in f may be crowd process duration, money spent, and some quality measure such as text length for a crowd task of text shortening.

4 The PPLib Programming Library

We have implemented each of the aforementioned core modules into PPLib, an extensible programming library that supports computer-crowd worker interaction.

The PPLib Programming Library is modeled after the generic human communication patterns described in speech acts [Searle 1969; Winograd and Flores 1986]. Comparable to Turkit [Little et al. 2010], one develops code that asks crowd workers questions and acts upon their answers. More complex interactions can be aggregated into reusable Crowd Processes relying on the principle of stepwise refinement [Wirth 1971] to keep code clean and simple.

Figure 11 shows the main components PPLib is comprised of and outlines how they work together.

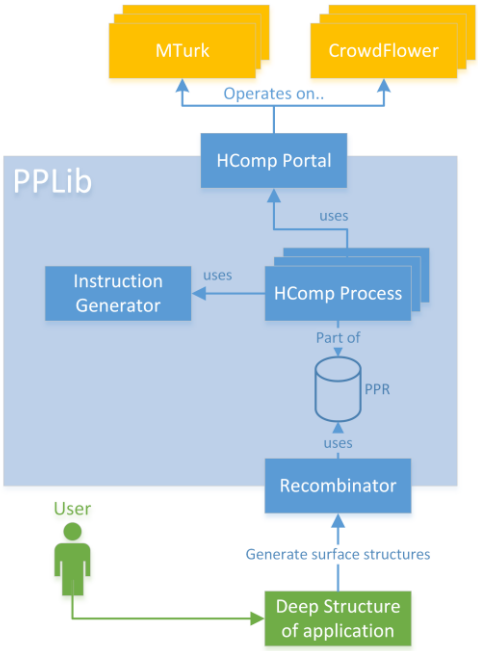

Figure 11: PPLib architecture

- **Recombinator**: Generates process surface structures based on the search space definition provided by the user's deep structure definition and constraints. While it is possible to use other components of PPLib directly, most users will only interact with the Recombinator.
- **HComp Process:** Central components within PPLib. They expose interfaces with lists of parameters that need to be supplied for the process to be instantiated.
- **PPR (PPLib Process Repository):** The collection of HComp Processes.
- **Instruction Generator:** Creates Instructions for crowd workers based on user defined recipes.
- **HComp Portal:** Abstracts communication with portals (e.g., MTurk) such that processes can be portal agnostic.

PPLib is written in Scala[18] and can therefore easily be integrated in any existing Java or Scala application using industry standard dependency managers such as SBT[19](Apache Ivy[20]) or Apache Maven[21].

[18] http://www.scala-lang.org/
[19] http://www.scala-sbt.org/
[20] http://ant.apache.org/ivy/
[21] http://maven.apache.org/

4.1 Human Computation Portals in PPLib

PPLib is built to be Human Computation Platform independent, i.e. programs using PPLib as an intermediary will run on any supported Human Computation Platform. PPLib comes with built-in support for CrowdFlower[22] and MTurk. Switching a process from one portal to another is very easy for PPLib applications and can be done by changing a parameter value of crowd process. One can even use workers from different portals in the same process and effectively have them interact with each other, transcending the borders posed by such portals.

Furthermore, PPLib is built in a way that makes it easy to add support for other existing and new platforms. Hence, support for other paid- and unpaid platforms such as UpWork[23] or Facebook[24] could be added.

4.2 Human Computation Processes in PPLib

A Human Computation Process in PPLib is any (Scala/Java) class inserted into the PPR using our ontology (see section 3.1) and implementing the defined interface methods.

To support long running transactions, PPLib includes reusable building blocks of the *crash and rerun* pattern introduced by [Little et al. 2010].

Among the most important functionalities of Crowd Processes is their sophisticated parameter system that supports the declaration of types and the definition of default values. Supported are all types of the Java/Scala language, including *other crowd processes*, which allows for the creation of *nested processes*. For example, many processes, require a DECIDE-type process to select the best option among multiple candidates.

Every process exposes its mandatory and optional parameters as well as their defaults and validity bounds in its meta-data. This list of parameters is later used by the Recombinator to generate recombinations, which then essentially finds all possible definitions for these declarations. In case of nested processes, recombination therefore amounts to a recursive traversal of such a dependency tree.

4.3 The PPLib Process Repository Content

We populated the PPLib Process Repository with crowd processes gathered from the crowdsourcing literature. We initialized it with 15 crowd processes out of which 5 do not nest any other process, and are therefore referred to as *basic building blocks*. As mentioned in section 3.1, there are two abstract classes (CREATE and DECIDE) that contain various enactable subtypes.

For the CREATE building blocks, we created two processes: (i) a *simple collection*, where the developer specifies the number of crowd workers that are asked to perform a certain task, and their outputs is returned in a list and (ii) a *collection with six-σ pruning*, as proposed by [Minder and Bernstein 2012].

DECIDE has four basic building blocks: (i) *majority voting* (ii) *majority voting with N winners*, where elements that have been selected at least N times win, (iii) *Beat-By-K voting*, as introduced by [Goschin 2014], and (iv) *Statistical Reduction voting*, as implemented in [Barowy et al. 2012].

[22] http://www.crowdflower.com

[23] http://www.upwork.com

[24] http://www.facebook.com

Besides the basic building blocks outlined above, the PPR also contains more complex process fragments, that allow nesting other modules from the PPR. The complex processes for CREATE initially implemented in the PPR are:

- *Iterative Refinement*: Iteratively asks crowd workers questions (using any CREATE element) and decides (using any DECIDE element) whether to use their answers or keep the previous state for the next iteration. The maximal number of iterations can be configured as a parameter.
- *Dual Pathway*: An adaption of [Liem and Chen 2011] with a DECIDE step.
- *CollectDecide*: Any CREATE process followed by any DECIDE process.
- *FindAndFix*: A DECIDE process with multiple winners followed by a CREATE Process.

The PPR can be extended by every user directly due to its Open Source nature. Extensions can be made available to the public by creating an open GitHub repository containing only the crowd process fragment's code and the SBT[19] instructions to reference the PPLib dependency.

4.4 Example: Using the PPLib Recombination Engine for Text Shortening

In this section we illustrate the use of the PPLib methodology by employing the PPLib framework for an actual use case. To that end, we will briefly walk through the four steps of the methodology and explain how they can be implemented in practice.

As an example use case, we are interested in shortening a book with 1000 pages. Our first step is to take a small sample of the book and use that sample to find the best crowd process to apply to the remainder of book.

STEP 1 – Intelligence: *Identifying the process deep structure*. After taking a sample of the book, we need to define the deep structure for the shortening process as well as a utility function. Algorithm 1 shows a possible implementation of the deep structure in Scala-code. It captures the core actions of text shortening: (i) dividing the sample into its paragraphs, (ii) obtaining a crowd process to shorten the paragraphs, (iii) running this process and then (iv) returning the result.

ALGORITHM 1. Process deep structure for text shortening

```
case class ShortNResult(text:String, costInCents:Int, durationInSeconds:Int) extends
    Comparable[ShortNResult] {

    override def compareTo(o:ShortNResult) = [..] // put utility function here to rank candidates
                                                  // according to outcomes
}

// The deep structure needs to implement PPLib's DeepStructure interface. SimpleDeepStructure
// can be used for deep structures that only require 1 recombined crowd process.
// The DeepStructure interface requires a definition of the input type applied to the deep
// structure as well as the output type. In this case, the deep structure uses a string
// input (text) and returns a ShortNResult as output (see above, captures resulting text, cost
// and execution duration)
class ShortNDeepStructure extends SimpleDeepStructure[String, ShortNResult] {

    override def run(input:String, b:RecombinedProcessBlueprints) : ShortNResult = {
      // (i) Split the text ("data") that was passed to this method into its paragraphs
      // (i.e., a list of patches, as required by inputType in obtainInstanceOfCrowdProcess)
      val paragraphs = IndexedPatch.from(input)

      // (ii) b will contain the blueprint for the recombined crowd process that we'll apply
      // for text shortening. The first step is to create an instance of that process
      val shortenerProcess = obtainInstanceOfCrowdProcess(b)

      // (iii) supply the list of paragraphs to the process and let the crowd shorten it
      val shortenedParagraphs = shortenerProcess.process(paragraphs)

      // (iv) return result (shortened Text) with its cost and duration
      ShortNResult(shortenedParagraphs,
                   shortenerProcess.costSoFar,
                   shortenerProcess.durationSoFar)
    }

    def obtainInstanceOfCrowdProcess(b: RecombinedProcessBlueprints) = {
      //we'd like to pass the list of paragraphs to this process and get a shortened list
      //paragraphs in return. Therefore the desired input and output-types for this process
      //are the same and equal a list of patches (which is the type of our paragraphs-
      //member variable)
      type inputType = List[Patch]
      type outputType = inputType

      //create an instance of the process according to its deep structure and supply the
      //desired types. Return this instance
      b.createProcess[inputType, outputType]()
    }
    //[..] search space definition from ALGORITHM 2 here
}
```

The *run*-method receives the recombined processes as well as the text to be shortened as parameters and uses them to specify the process deep structure. (i) Since most processes in the PPR operate on a built-in data structure called *Patch*, the first step of the deep structure is to create patches from the supplied text sample. A *Patch* is an abstract entity that is used for any item that needs to be processed

using PPLib. In this case, it represents a paragraph and its position in the text. (ii) The deep structure then creates an instance of the crowd process it will use and (iii) executes it, by calling the *process*-method and passing the prepared data structure with the list of patches resembling the paragraphs. (iv) Lastly, the result structure (ShortNResult) is constructed using the cost and duration of the process as well as the outcome of the process, i.e. the shortened paragraphs.

STEP 2 – Design: Defining parameter candidates. After defining the deep structure, we now need to instruct the Recombinator correctly, such that it can generate surface structures that apply to our use case of shortening paragraphs using crowds as shown in Algorithm 2. (i) In a first step, we define the applicable basic type from the PPR (CREATE / DECIDE) and the input and output types. (ii) We then define the crowd worker instructions that are to be used in the target process and (iii) bind these instructions to all generated processes. (iv) Our last step is to put everything together in a search space definition and return it.

Note that this all takes place in the same class as the one that we've used in step 1.

ALGORITHM 2. Defining the search space for the Recombinator

```
import ch.uzh.ifi.pdeboer.pplib.process.recombination.RecombinationHints._

class ShortNDeepStructure extends SimpleDeepStructure[String, ShortNResult] {
    // ... run method as shown in algorithm 1
    override def defineSimpleRecombinationSearchSpace = {
        // (i) Our target process should be a Create-type process that takes any descendant
        // of a list of patches as input (in Scala notation: "_ <: List[Patch]") and output.
        // This includes specializations of lists and/or the Patch-class.
        type targetProcessType = CreateProcess[_ <: List[Patch], _ <: List[Patch]]

        // (ii) The most important restriction on the search space is to define the crowd
        // task. By using "InstructionData" we also specify how a question gets rendered
        // to the crowd. On a process, an InstructionGenerator (IG) will then phrase
        // instructions in a way that fits the process. For example, in a majority vote
        // process, the IG will phrase a question out of this base data to pick the best
        // shortened paragraph while paying attention to grammar and text-length.
        val instructions = RecombinationHints.instructions(List(
            new InstructionData(actionName = "shorten the following paragraph",
                detailedDescription = "grammar (e.g., tenses), text-length")
        ))

        // (iii) Specify that we'd like to use the default hints for all building blocks that
        // the Recombinator processes. It would be possible to target specific building
        // blocks and supply different hints to them.
        // The default hints are just the instructions specified above.
        val hints = RecombinationHints.create(Map(DEFAULT_HINTS -> instructions))

        // (iv) Construct the search space and return it
        RecombinationSearchSpaceDefinition[targetProcessType](hints)
    }
}
```

(i) We first define the target type from the PPR that we'd like to use (CREATE). This Create Process should be able to work on a list of patches as input as well as a list of patches as output.

(ii) We then define the instructions that are used to generate questions to crowd workers. PPLib can generate instructions automatically based on provided core formulations and adapt them to the kind of process that uses the instructions (CREATE / DECIDE). For our initial experiments, we have used a simple instruction generator, that inserts different predefined text for CREATE and DECIDE to make a question out of the core instruction data. It is possible to extend PPLib with other, more complex

instruction generators. A more detailed explanation on instruction generation is provided on our GitHub repository[25].

(iii) After specifying the formula to generate instructions for crowd workers, we use it as the only default parameter and (iv) construct the search space using instructions and the target type definition.

STEP 3 – Choice: Recombination and Auto-Experimentation. This step brings it all together: In the executable Main class of our PPLib project (see Algorithm 3), (i) we first create an instance of the deep structure as specified in step 1. (ii) The deep structure is then passed to the Recombinator who uses it to generate surface structures of our process, i.e., its concrete implementations. (iii) Lastly, the surface structures are supplied to the Auto-Experimentation Engine and executed there. The Auto-Experimentation Engine uses the utility function as defined in step 1 to determine the best performing process among all the surface structures.

ALGORITHM 3. Recombination and Auto-Experimentation

```
object ShortnText extends App {
     val text = "shorten me [..]" //text that needs to be shortened. E.g., could be loaded from PDF

     val deepStructure = new ShortNDeepStructure //(i) instantiate step 1 & 2 from above

     //(ii) use the search space defined in the deep structure to generate the surface structures
     val surfaceStructures = new Recombinator(deepStructure).recombine()

     //(iii) create AutoExperimentationEngine and let it run one iteration on all surface structures.
     //Note, that one can easily filter the surface structures before running Auto-Experimentation.
     val results = new AutoExperimentationEngine(surfaceStructures).runOneIteration(text)

     //use utility function to find the process with the best result. Alternatively, one can also
     //export the result and pick the best one using one's ranking tool of choice.
     val bestProcess = results.bestProcess

     println(s"The best process is $bestProcess")
}
```

5 Evaluation

In this section we evaluate our claims by showing the usefulness of the PPLib methodology and library through the implementation of two non-trivial but typical crowd-processing examples – text translation and text shortening.

By the time of submission of this paper, PPLib was used to process more than 19'400 micro tasks involving more than 900 unique workers (by Turker ID). All the micro tasks submitted for this evaluation were restricted to experienced [26] workers from the United States. Crowd worker compensation depended on the time allotted for a crowd task[27].

The validation of PPLib proceeds by applying it on two well-known crowdsourcing problems (text translation and test shortening) and evaluating its results.

5.1 Text translation

In order to evaluate the usefulness of the PPLib methodology and library, we have applied it to find crowd process suitable to translate a German text to the English language. After recombination and

[25] https://github.com/uzh/PPLib/blob/master/docs/instructiongenerator.md

[26] Workers with less than 4% rejected hits, and more than 4000 approved hits

[27] We care about crowd workers and are happy to report to have a flawless 5/5 score on all dimensions on Turkopticon at the time of submission

Auto-Experimentation, the result of each (recombined) crowd process is compared to a benchmark translation along three dimensions to establish a ranking in terms of suitability. A professional translator finds the results of the top-5 crowd processes to "very similar" to the benchmark text.

We used an article of the Swiss bank UBS' news portal available in both, German and English, with the English version serving as our benchmark[28]. The article consists of 209 words in 16 sentences.

The algorithm's deep structure was inspired by the pattern used in [Minder and Bernstein 2012] and first translated the article from German to English using Google Translate as a baseline. The resulting text was then split into its paragraphs (2-3 sentences each), which were then refined in a CREATE-type task by crowd workers. Afterwards, the refinements were collected and aggregated into a full article, based on their original positions.

For our experiments, we restricted parameter recombination to the testing of two worker counts (i.e., 5 and 7 for CREATE steps, 3 and 5 for DECIDE steps) and did not vary other parameters such as *K* for Beat-By-K. This resulted in the 8 basic building blocks as detailed in Table 4:

4 DECIDE variations	4 CREATE variations
○ MV3: Majority voting with 3 voters ○ MV5**:** Majority voting with 5 voters ○ BV: Beat-By-K Voting with a maximum of 20 votes cast ○ SV: Voting with Statistical Reduction with a confidence level of 85% and a hard maximum of 20 votes cast	○ SC5**:** A simple collection made by 5 workers ○ SC7**:** A simple collection made by 7 workers ○ PC5: A collection with six-σ pruning made by 5 workers ○ PC7: A collection with six-σ pruning made by 7 workers

Table 4: Building block variations used in the Translation experiment

The process recombination was unrestricted and lead to 6 nested processes, as shown in Figure 12 Each nested process was recombined with all applicable building blocks from Table 4. Figure 12 indicates the number of applicable building blocks for each nested process in brackets (DECIDE is orange, CREATE is green, nesting is black)

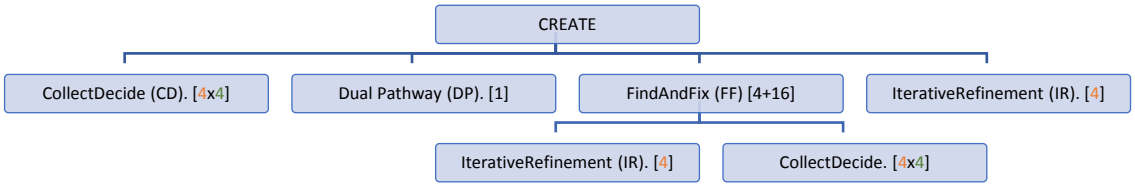

Figure 12: Nested Processes Used in Translation Experiment (number of applicable building blocks for each nested process in brackets, where DECIDE is orange, CREATE is green, nesting is black)

The search space definition, therefore, mostly exploited process recombination and lead the Recombinator to generate 41 surface structures (From left to right in Figure 12: CD (16) + DP (1) + FF(IR(4) + CD(16)) + IR(4) = 41).

[28]http://www.ubs.com/global/en/about_ubs/about_us/news/news.html/en/2014/12/29/consumption-indicator.html also available on our GitHub repository

After running all resulting 41 candidate processes once, we obtained 41 translations. To establish the fitness for a crowd process, we defined the utility function to compare the resulting text of a crowd process candidate to the benchmark along 3 dimensions:

- *Readability*: How easy to understand are the resulting texts?
- *Adequacy*: The content of the reference translation is adequately represented by the translation algorithm.
- *Fluency*: The translated text should be grammatically correct and fluent.

Readability was automatically rated using the Flesch-Kincaid Grade Level [Kincaid et al. 1975] (normalized to values between 1-5), while adequacy and fluency were rated by 10 translation experts[29] recruited on UpWork[30] on the 5 point Likert scale. Specifically, we took the 41 resulting translations and randomly assigned them to the experts assuring that we would get 10 expert ratings per text (average rating μ shown in in Table 5). The utility function for a crowd process was defined as the sum of the ratings in these three dimensions.

Based on the results of this utility function, a ranking of the suitability of the 41 initial crowd process candidates for the problem of text translation could be established.

We additionally hired a professional translator to rate each crowd processes in terms of its adequacy and fluency in order to validate PPLib's results. Table 5 lists the top 5 crowd process candidates and their ratings by both UpWorkers and the professional translator. For comparison, it also includes ratings for the benchmark.

Recombination	Adequacy (1 to 5)		Fluency (1 to 5)		Readability (1 to 5)	Utility value (3 to 15)
	μ	Professional	μ	Professional		
FF-PC5-BV	4.8	5	4.6	5	4.2	13.6
CD-SC5-SV	4.9	5	4.5	5	4.0	13.4
FF-SC7-MV3	4.8	5	4.3	5	4.2	13.3
FF-SC7-BV	4.7	5	4.5	5	4.0	13.2
CD-PC7-MV5	4.5	5	4.7	5	4.0	13.2
Benchmark	4.8	5	4.8	5	5.0	14.6

Table 5: Ratings of the 5 top-performing recombinations for text-translation[31].

The Kendall-Tau correlation between the professional translator and the average rating of the expert crowds members was high (Adequacy: ~0.54; Fluency: ~0.59). The quality of the translations created by the top-5 processes were rated as "high" by a professional translator.

The crowd process reaching the highest overall score of 13.6 consists of a *Find-step*, followed by a *Collection with six-σ pruning* whose best refinement is chosen in a *Beat-By-K voting process*. This particular

[29] Excellent average grade on eLance (Higher than 4.5 points out of 5) as well as the UpWork Qualification for the translation of German to English

[30] www.upwork.com

[31] Note that the ratings of the professional do not vary in the table, because the table shows only our 5 top-performing processes. The professional ratings indeed range from 3-5 over all processes

composition of a crowd process has not been proposed in literature before. In fact, most of the processes we identified as performing particularly well in the environment of text-translation have not previously been proposed.

It is interesting to note, that well-known design patterns such as *Iterative Refinement* or *Dual Pathway* did not excel at this task. This indicates that the systematic exploration of the design space in text translation using Process Recombination provides reliable improvements over the baseline of picking a well-known pattern among the ones proposed in the literature.

5.2 Text shortening

Shortening texts without losing information is a central task in editing and has de-facto become one of the common baseline problems to evaluate crowd processes on. For our second experiment, we have therefore executed PPLib for the problem of text shortening and compared the outcome of PPLib with the baseline design pattern for text shortening: Find-Fix-Verify [Bernstein et al. 2010a].

The code for the text shortening evaluation resembles Algorithm 1-3 in section 4.4.

As example text, we used an English article from the popular news platform *the verge* with 1444 characters[32] and split it up into its individual paragraphs. Each paragraph was then processed in a CREATE-type task, asking crowd workers to shorten the paragraph. We were eager to see whether the process that performed best in the translation task would excel in this problem context too; we therefore reused the same selection of abstract processes as in preceding section, 5.1. As a utility function to rank crowd processes, we used text length (shorter text = better).

All of the recombined candidates were executed four times on different work days of the week in order to evaluate their stability. We measure crowd process stability by the standard deviation of the ranked performance across the four iterations. The 5 processes with the highest median utility, i.e. the shortest median text length, were considered the main outcome of the PPLib system. These were evaluated by asking 21[33] UpWorkers to rate the adequacy of their shortest text on a 5-point Likert scale. Note, that in text shortening, shorter texts often mean a loss of detail; which reduces adequacy. Using the shortest text coming out of the 4 executions of a crowd process therefore represents the most cautious measure of crowd process adequacy.

Table 6 shows the 5 candidate processes with highest utility. The best process among these 5 shortened the article to 82% of its length in average across the 4 repetitions.

[32] http://www.theverge.com/2015/1/8/7517361/google-getting-ready-to-sell-auto-insurance-and-maybe-buy-coverhound also available on our GitHub Repository

[33] 20 workers + 1 backup worker, in case someone wouldn't have provided his answer on time.

Recombination	Text length (characters)		Adequacy (1 to 5)		Cost (US Dollars)	
	μ	σ	μ	σ	μ	σ
CD-SC5-MV5	1198.0	99.4	4.1	0.7	$4.25	$0.31
CD-SC5-MV3	1202.5	54.9	2.9	1.2	$3.45	$0.42
CD-SC5-BV	1278.5	82.9	3.3	1.1	$3.08	$0.38
CD-PC5-MV5	1281.0	68.3	4.1	0.9	$4.00	$0.29
IR-BV	1286.0	53.7	2.7	1.5	$5.28	$0.56

Table 6: Top 5 candidate crowd processes for text shortening according to the results of their utility function (minimizing text length)

Among the generated crowd process candidates were variations of Find-Fix-Verify [Bernstein et al. 2010a], with different numbers of workers for Fixers and Verifiers. The best-performing Find-Fix-Verify process shortened the text to an average of ~90% of its original length and was ranked 6th among our recombinations. In our experiments, however, we found the 5 recombinations shown in Table 6 to consistently outperform Find-Fix-Verify in terms of text length.

In summary, our approach also performs well when applied to text shortening: the systematic exploration and experimentation of the process design space yielded 5 stable recombinations that performed reliably better in terms of text length than the baseline provided by Find-Fix-Verify.

5.3 Comparing the evaluations: Text Translation vs Text Shortening

Among our main claims is, that a processes' performance varies in different applications. If true, the strategy of reusing a preferred design pattern for any new problem would lead to varying results, which might sometimes be disappointing. In this paper, we present a possible remedy to this problem, the automatic exploration of the design space of alternatives when encountering a new application domain. Since we essentially evaluated the same crowd process candidates for both real-world problems described in the preceding sections 5.2 and 5.1, a comparison between individual crowd process performance in the two problems would allow to explore this central claim. More specifically, we compared the rank of the top-50% most stable text-shortening processes in terms of text length with their respective rank in the translation experiment. Rankings are defined by crowd process (mean-) utility. The Kendall's Rank Correlation (τ_K) was τ_K =0.04 with P=0.82 indicating the complete absence of a relationship between the rankings of the two data sets.

As a second step, we compared specific building blocks performance in the two experiments. To this end, we computed Kendall's Rank correlations for all recombined processes containing a specific building block. Some building blocks were extremely unstable, such as Majority Vote (τ_K=-0.05; P=0.88) or Collect-Decide (τ_K =-0.14; P=0.60), both giving evidence for an absence of a relationship between the performances in either setting.

Based on these findings, we contend *what works in one case might not work in another*. An approach purely based on best practices may therefore not yield perfect results.

6 Discussion

PPLib well applicable to scenarios, where a task requestor has many predefined tasks that need to be completed by crowd workers in a process-like fashion. The PPLib approach imposes an upfront cost for finding a crowd process performing well for a tasks at hand. It is worth taking the PPLib-route, if

the efficiency gains the PPLib-nominated crowd process achieved over all tasks are higher than the upfront cost, which is likely dependent of the problem-domain and the number of tasks to be solved.

Besides Recombination and Auto-Experimentation, PPLib also introduces a simple, other advantage: It provides implementations for many well-known crowd patterns out of the box and doesn't require a potentially expensive programmer to develop a crowdsourcing process all by herself.

The cost of using PPLib depends on the sample size of the tasks a requestor would like to complete and on the task complexity. A PPLib user specifies the money spent on every query sent out through PPLib and can, therefore, calculate the total expenses for processes with fixed query counts and specify a cost-ceiling for processes with dynamic query counts (e.g., Beat-By-K). We provided the functionality to do this automatically in PPLib by calling the method *getCostCeiling()* on the Recombinator after the search space definition.

7 Limitations & Future Work

Generating all possible combinations of crowd process candidates leads to a search space that grows exponentially with every additional parameter to be recombined. A complete exploration of the search space through Auto-Experimentation is therefore often not possible with limited funds and limited (human/computational) processing power. Furthermore, in some domains, even the Recombination of all candidates is impossible, as it may lead to an infinite number of processes (e.g., due to continuous valued parameters). Hence, both the design space exploration via recombination and the Auto-Experimentation engine need to be improved: (i) The Auto-Experimentation engine should be extended to consider the robustness of results (i.e., the distribution of result quality), which is what we did manually in our evaluation. (ii) The current engine should be extended with findings from active sampling techniques, so as to avoid blindly sampling every process *i* times, and, instead, choose samples strategically. This is especially important in situations that face a certain experimentation budget [Saar-Tsechansky and Provost 2001; Sheng et al. 2008]. (iii) Integrating the Auto-Experimentation engine and the Recombinator will allow to interleave (and trade-off) the exploration of the design space (i.e., recombination of new processes) with the exploitation of candidate recombinations (i.e., experimentation). We hope that the combination of these improvements may extend our approaches' performance when exploring solutions in high-dimensional domains with a limited budget. (iv) The power of the recombination engine is dependent on the abstract processes included in the Process Repository and the deep structure devised by the user. Akin to HTN-planning, which is limited by the plan grammar, our approach is limited by the diversity of the PPR to explore the possibilities for implementing the deep-structure. Hence, we built the PPR to be easily extensible.

Another important consideration is that the quality of the results of PPLib is dependent on the appropriateness of the user-defined utility function. Additional research is needed to explore the suitability of utility functions to different tasks.

As always in such settings, our experimental results are limited by a number of threats. First, we only ran a limited number of texts, which potentially limits the generalizability of our results to other texts. However, when taking into account with the findings of [Minder and Bernstein 2012], who ran a larger variety of texts through a much smaller number of manually recombined processes, we believe that our newly proposed methodology, with its automated recombination, approach stands and that our findings should be stable also for other texts. Second, in a similar vein, we could not run experiments with all possible environments and crowdsourcing tasks. But, we believe that our selection of two typical tasks provides sufficient evidence that our approach merits great potential for generalizing to other applications. Nevertheless, future experiments will, therefore, further explore PPLib's

generalization to other classes of tasks. Last but not least, we experienced some variability of experimental results, due to the variability of worker performance. In the text shortening validation, we found that some processes did not effectively control for worker variance and, therefore, yielded results of highly varying quality. Adhering to standard practice, we tried to address this issue using appropriate experimental procedures and statistical tests.

8 Conclusion

Probably the most popular way to do crowd process design today is reusing various proposed design patterns from literature, which is essentially a best practices approach. Especially in a situation, where one needs to run a specific crowd process repeatedly, it is very important for this crowd process to be as efficient and effective as possible – something we believe cannot be achieved with the currently established approach.

In this paper we introduced PPLib, both as a methodology and as a system implementing this methodology. PPLib leverages a repository of crowd process fragments as well as the notions of Recombination and Auto-Experimentation to systematically explore the design space of possible processes by generating and trying alternatives for Human Computation tasks.

We evaluated our system threefold: (i) *Text translation*, where English-speaking crowd workers were asked to refine a machine translation of a German news article. Our system autonomously generated various recombinations for the human computation task, which were then supplied to the Auto-Experimentation engine and yielded translations coming close to the benchmark of a translation produced by professionals. Our recombination engine unearthed various processes that have not been proposed before, among them the process whose performance in the environment of text translation was best in terms of adequacy, fluency and readability.

(ii) *Text shortening*, where our recombination mechanism generated 41 process candidates. Among the recombined processes were four variations of the popular Find-Fix-Verify pattern proposed by [Bernstein et al. 2010a]. While the results indicated that Find-Fix-Verify performs efficiently, its text-length consistently exceeded the shortened texts of five other generated processes in four iterations.

(iii) In a comparison of both of our experiments, we could conclude that high-quality crowd processes as well as crowd process fragments do not generalize between settings, indicating the lack of silver bullets in the crowdsourcing domain, which lead us to doubt the best practices approach common currently.

Given these findings, we believe that our approach, based on crowd process Recombination and Auto-Experimentation, is a first candidate for systematic crowd process design – a relatively new area that warrants further exploration.

9 Acknowledgements

We would like to thank Patrick Minder for his support in the evaluation of the crowd translations. We are also thankful for help of Dan Lowenthal, Tobias Grubenmann, Michael Feldman and the anonymous reviewers of this paper, whose input was central for improving the paper. This work was supported in part by the Swiss National Science Foundation (SNSF- Project: 200021-143411/1).

Efficiently Identifying a Well-Performing Crowd Process for a Given Problem

This chapter is based on a paper that has been published in the proceedings of the conference **Computer Supported Cooperative Work (CSCW)** [de Boer and Bernstein 2017]. We presented a short version of it at the Collective Intelligence conference 2016 as a poster [de Boer and Bernstein 2016a].

Efficiently Identifying a Well-Performing Crowd Process for a Given Problem

ABSTRACT

With the increasing popularity of crowdsourcing and crowd computing, the question of how to select a well-performing crowd process for a problem at hand is growing ever more important. Prior work casted crowd process selection to an optimization problem, whose solution is the crowd process performing best for a user's problem. However, existing approaches require users to probabilistically model aspects of the problem, which may entail a substantial investment of time and may be error-prone. We propose to use black-box optimization instead, a family of techniques that do not require probabilistic modelling by the end user. Specifically, we adopt Bayesian Optimization to approximate the maximum of a utility function quantifying the user's (business-) objectives while minimizing search cost. Our approach is validated in a simulation and three real-world experiments.

The black-box nature of our approach may enable us to reduce the entry barrier for efficiently building crowdsourcing solutions.

1 Introduction

In recent years, human computation attracted the interest of the CSCW community [Cheng and Bernstein 2015; Zhu et al. 2014; Kulkarni et al. 2012], since the computer-mediated coordination of crowds can be seen as an example of computer-supported cooperative work. The question of identifying the optimal coordination structure ("*crowd process*") for a problem at hand caused prior work [Zhang et al. 2013; Dai et al. 2013] to use methods from the field of optimization. These approaches rely on probabilistic modelling of (sub-)tasks of the problem or of its workers. The drawback of such white-box optimization based methods is, that modelling by the end-user can be high-effort and error prone. Additionally, practitioners who do not possess probabilistic modelling skills are widely unable to use them. For such practitioners, it is therefore often unclear how to efficiently design an inexpensive and accurate crowd process for a task at hand. Many resort to a manual approach, where they may rely on the various design patterns that have been proposed in literature. Unfortunately, it is often unpredictable which pattern works well for a given problem, since their performance, and hence their applicability, varies by problem setting [de Boer and Bernstein 2016b].

This is costly and time-consuming. PPLib [de Boer and Bernstein 2016b] addresses this problem. It is a method and tool for systematic crowd process design, which, given a problem definition, automatically derives a set of suitable candidate crowd processes. However, after arriving at a set of candidates, PPLib's approach to identifying the best suited crowd processes among them is very expensive. It entails the (repeated) execution of the whole set of applicable candidate crowd processes for their evaluation through a user-defined utility function. The goal of this paper is a *widely accessible method to drastically reduce the cost of crowd-process selection* (i.e., the selection of a well-performing process from a set of candidates), such that the PPLib methodology can be employed as a more efficient way to systematically design crowd processes. To that end we propose to employ global (black-box) optimization for what is called Auto-Experimentation. The goal is to approximate the optimal parameterization of a crowd process – according to a given utility function. Parameters in this optimization include the kinds of patterns employed (e.g. Find-Fix-Verify [Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell 2015]), the coordination processes utilized [Thomas W. Malone et al. 2003], the crowd market to use (e.g., Mechanical Turk or CrowdFlower), as well as simple parameters such as the number of workers

involved in a majority vote. Specifically, we propose the adoption of Bayesian optimization [Mockus 2012], as it has been shown to find near-optimal solutions in non-deterministic settings whilst minimizing the number of samples (or actual process executions in our application). The key contributions of this paper are:

- The use of black-box optimization for crowd process selection
- An efficient selection method robust to the non-determinism of crowd processes.
- A publicly available Open Source system [34] implementing this new Auto-Experimentation strategy.

2 Related Work

This paper continues and combines research streams in the fields of human computation, Auto-Experimentation, and global optimization.

2.1 Human computation patterns

A popular approach to scale the process of solving large problems in the field of crowdsourcing, is to harness smart task decomposition. For example, a crowd process translating a book could divide the book into its paragraphs, for each paragraph concurrently ask a crowd worker to provide a translation, and then compose a translated version out of all proposed paragraphs.

If one were to use the translation process outlined above, one would get a book with paragraphs of varying quality – some of which may be unsatisfying. Therefore, various *design patterns for quality assurance* have been proposed: For instance, *Find-Fix-Verify* [Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell 2015] asks crowd workers to first select an item that needs improvement within a list of items and then employs multiple crowd workers to suggest alternatives to this item followed by multiple crowd workers selecting the best alternative in a majority vote. If the list of items to be worked on is large and inter-dependent, *Context-Trees* could be used instead [Verroios and Bernstein 2014]. More generally, [Malone et al. 2010] describe the *Contest* pattern as asking multiple crowd workers to propose an answer to a given question, followed by a majority vote to nominate the best answer. The *Iterative Refinement* [Little et al. 2010] pattern iterates such contests until a majority vote decides that the improved version is not better than the version of the previous iteration. *Statistical methods for quality assurance* aim at weighting answers by crowd workers trust worthiness [Inel et al. 2014], or estimate the number of distinct crowd answers needed for a given problem [Barowy et al. 2012; Livshits and Mytkowicz 2014; Ertekin et al. 2012].

2.2 Auto-Experimentation for Crowd computing

How can designers determine which design pattern is ideal for a problem at hand? Auto-Experimentation Engines, systems that autonomously plan and run the evaluation of a given set of hypothesis', may be adopted towards this cause. They have been used successfully in biology [King et al. 2009], machine learning [Serban 2010a], and, most recently, crowd computing [de Boer and Bernstein 2016b]. Specifically, de Boer and Bernstein proposed PPLib, a method and tool capable of automating large parts of crowd process design by viewing crowd process suitability to a given problem as a hypothesis to be answered through Auto-Experimentation.

In order to employ PPLib, a crowd process designer needs to first specify the problem deep structure, i.e. the most abstract way of formulating a problem [Chomsky 1965], using operators available through

[34] Integrated into https://github.com/uzh/PPLib

PPLib's Process Repository (PPR). PPLib will then automatically recombine existing crowd process fragments (some of which are part of the design patterns listed above) to arrive at an often large set of candidate processes suitable for the given problem definition [Bernstein et al. 1999]. When supplied with a user-defined utility function, PPLib's Auto-Experimentation module can then be employed to automatically find the best crowd process among these candidates by executing them and assessing their result desirability with the provided utility function. This function can incorporate various parameters to grade an individual crowd process-execution, e.g. crowd process result, cost or run time. Figure 13 shows the general workflow of using PPLib.

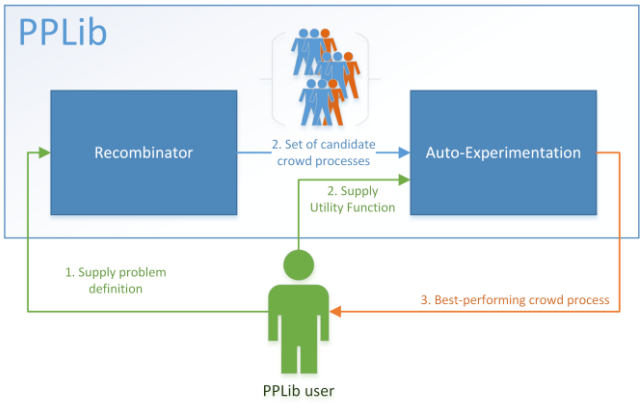

Figure 13: The PPLib approach

In PPLib, Auto-Experimentation works by taking a sample of the task's input data (e.g. if the task is to translate a book, a sample could be a few randomly selected pages of the book), testing the performance of a candidate crowd processes on the sample, and evaluating its results (along with other parameters, such as cost and duration) using the user-defined utility-function. The best crowd process is then selected as the one with the highest result of the utility function. PPLib's Auto-Experimentation module has initially been proposed with a single strategy: Naïve Auto-Experimentation (NAE), which simply executed all candidate crowd processes and compared their (average) results. By default, the NAE-strategy executes each process candidate 4 times, and returns the process with the highest median utility of its results. It repeats a candidate process' execution, because crowd processes are non-deterministic, i.e. their performance characteristics may vary with each execution due the cognitive- and error diversity of human actors [Bernstein et al. 2012] or other stochastic elements of the process. In the following, we denote the number of repetitions used in a NAE variant by appending it, e.g. NAE-4 for the default setting. This naïve approach to process evaluation presented by NAE is very expensive and reduces the usefulness of the PPLib method to settings where the costly experimentation can be amortized (e.g., over many runs or few executions with a very high payoff).

2.3 Optimization for workflow selection in Crowdsourcing

The naïve approach to crowd process selection is expensive. There has therefore been interest to cast workflow selection to an optimization problem. Zhang et al. [Zhang et al. 2013] compose crowd processes from generic tasks that resurface in different combinations. Using a manually created (probabilistic) model for task performance, one can then estimate the utility of all crowd processes compiled of these generic tasks. This potentially enables finding optima even for large sets of candidate workflows. However, their work is limited to tasks with finite outcomes such as Majority Votes and currently does not support open-ended outcomes such as writing texts. [Dai et al. 2013] use a decision-

theoretic approach with Partially-Observable Mixed Markov models to capture crowd worker accuracy for repeated labelling in face of noisy workers.

Both approaches face challenges in practical applications as they require to manually build (probabilistic) performance models for tasks (Zhang et al) or for workers (Dai et al). More generally, these so-called white-box approaches are tied to specific crowd processes and would need to be adapted to support further processes. Wider adoption of such techniques might therefore be constrained, as probabilistic modeling can be high effort and error prone. To address this challenge, [Weld et al. 2011] suggested to infer such models using machine learning techniques in a system named Clowder. Although promising, an evaluation and detailed description of Clowder are still pending.

Our method, in contrast, sidesteps the need for modeling by employing a pure black-box approach. This has the advantage that our approach is not tied to specific crowd processes and can therefore be applied to many problems out-of-the-box. The lower entry barrier aims to open optimized crowd process selection to a wider audience. However, the main disadvantage of our approach when compared to white-box methods, is that it only approximates optima rather than being guaranteed to find optima. In practice, this means that our approach nominates good processes as opposed to always identifying the best one. The validation section will quantify this effect.

2.4 Bayesian Optimization

In the realm of optimization, one aims to find either the minimum or the maximum value of a mathematical function, called *objective function*. Bayesian Optimization is an optimization technique which is especially useful when applied to objective functions that are expensive to evaluate and for which the closed-form expression is not available (i.e. no derivative is available). It has been applied to a plethora of problems, e.g. classifier hyper parameter optimization [Thornton et al. 2013], robotics [Metzen et al. 2015] and the configuration of a distributed computing framework [Fischer et al. 2015]. The evolution of Bayesian Optimization to a general-purpose optimization strategy was supported by multiple high-quality frameworks implementing it, among them Spearmint [Snoek et al. 2012], TPE [Bergstra et al. 2011] and SMAC [Hutter et al. 2011].

Bayesian Optimization is based on two components: the *surrogate function*, approximating the (black box) objective function, and the *acquisition function*, used to determine where to sample next. The algorithm iterates the steps of obtaining the next point to sample by maximizing the acquisition function, followed by the costly sampling of the objective function and updating the surrogate with the resulting posterior. The acquisition function aims to tradeoff between exploration (i.e., where the surrogate's estimate is particularly uncertain) and exploitation (i.e., where the objective function is expected to be particularly high). Various strategies for the acquisition function have been proposed, such as Probability of Improvement (PI), Expected Improvement (EI), or GP Upper Confidence Bound. In this paper, we use EI, since it offers a good tradeoff between exploration and exploitation. For a more in-depth introduction, we would like to refer the reader to the tutorial by [Brochu et al. 2010] and Jonas Mockus' thorough book on Bayesian Optimization [Mockus 2012].

3 Method: Process Design For Optimization

The goal of this paper was to find a method to efficiently identify a particularly fitting crowd process for a given problem from a set of candidate crowd processes. A simple, albeit inefficient, approach would be to execute all candidates and rank them according to how well their result fits the expectation of the task requester. To formalize her expectations, a task requester could specify a *utility function* μ, which takes all relevant parameters of an executed crowd process such as input/cost/duration and returns a numeric utility of how well a given crowd process execution fits the task requesters business

objectives. Ways of quantifying these business objectives for μ can be borrowed from the field of decision analysis, e.g. judgment bootstrapping [Dawes 1971].

Additionally, we introduce an *execution function* σ, whose input is an executable *crowd process* c. σ executes the supplied crowd process c and uses μ to evaluate c's results. By definition, σ's value is highest for the crowd process best catering to the task requesters goals defined in μ.

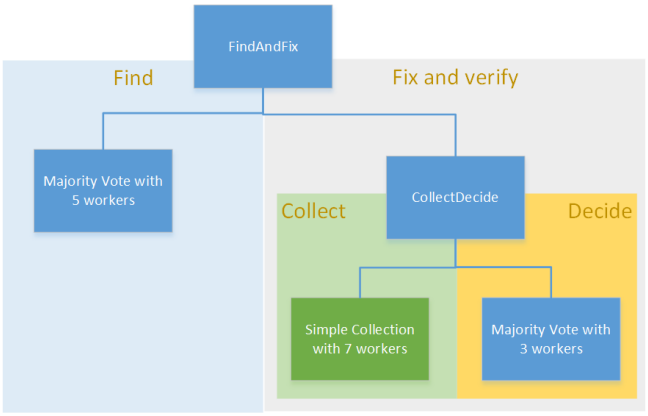

Figure 14: Tree-based visualization of the composition of an example variant of Find-Fix-Verify in PPLib [de Boer and Bernstein 2016b]

Concrete crowd processes (or processes in general) can be seen as a composition of crowd process fragments as well as a specification of the fragments' possible parameters [de Boer and Bernstein 2016b]. Figure 14 shows the composition of the popular Find-Fix-Verify [Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell 2015] pattern from fragments, where, for example, the additional parameter for the number of crowd workers finding a relevant element has been set to "5". Hence, every concrete crowd process c is defined by the values of its *parameters* P_c that specify the process fragments used and the concrete values for the fragments' parameters. For example, in Find-Fix-Verify (Figure 14), P_c includes categorical parameters such as a coordination mechanism (e.g. "Majority Vote") and numerical ones (such as the above-mentioned number of votes). Note that the size of P_c varies as some crowd processes may require more parameters than others. P_* denotes the set of all parameters known to the system, whereas each P_c therefore contains a subset of P_*. Each P_c contains one parameter that determines the root of the process composition. The process defined in Figure 14, for example, specifies *FindAndFix* as its root in its corresponding parameters. The *composition function* φ is then used to compose a crowd process based on a given subset of parameters from P_*. Essentially, given φ, the domain of P_* defines the *process design space*: a multi-dimensional space of possible process designs for a given problem. φ is generic, as it can construct any process specified via the parameters in P_c.

An optimizer can then be instructed to explore the process design space by varying the parameters to find the maximal value of σ. More formally, in each iteration i, an optimizer systematically choses a combination of parameter values V_i for the parameters from P_* relevant in iteration i and executes $\sigma(\varphi(V_i))$. Note, given that V_i also determines the structure of the process via some parameters. It stops iterating when it cannot reliably improve the maximally observed utility value or when a predefined number of iterations is reached. It then returns the parameter set that resulted in the highest value of σ as the best process for the user's μ. As an optimizer, we use Bayesian Optimization [Brochu et al. 2010], where the objective function is modelled using a Gaussian Process (GP). Intuitively, a GP is a distribution over functions, which means that each data point on a GP (i.e. *a crowd process' utility*) is

modelled by a Normal Distribution with its own variance. We use this property to encapsulate both, the optimizers uncertainty about the data point and the corresponding crowd process' performance's variance. Hence, the crowd processes non-determinism (i.e., its performance variance) is handled naturally by the GP. As the optimizer runs, it uses EI (see related work) to tradeoff the GP's current variance of different data points in the search space and exploitation of the acquired knowledge.

4 BOA: Bayesian Optimized Auto Experimentation

Based on the framework introduced in the past section, we now introduce its implementation within a new module in PPLib called BOA. For a step-by-step tutorial we would like to refer to BOA's user manual[34] instead.

4.1 Overview

PPLib first (A) derives a set of candidate crowd processes based on the problem deep structure via recombination. The set of candidate processes is then used (B) to inform the optimizer about the parameter space P_* and its bounds. Once started, the optimizer (C) iteratively requests samples of the crowd process design space for a given set of parameter values V_i. (D) Once the optimizer reaches convergence, the most reliable process maximizing the users μ is returned. We will now visit each of these steps in detail.

4.2 (A) Problem Definition & Recombination

Following the PPLib-approach (see Figure 13), the deep structure of the crowd sourced problem needs to be described using abstract operators. PPLib's recombination mechanism can then be employed to automatically derive a set of candidate crowd processes for the supplied problem definition. The result of this procedure is a set of all valid V_i's.

For a more in-depth explanation of Recombination, deep structures and the PPLib-method in general, we would like to refer to [de Boer and Bernstein 2016b], since this paper's focus is only on Auto-Experimentation.

4.3 (B) Configuring the optimizer

Given the set of all valid V_i's, the optimizer can be instructed about the dimensions and parameter bounds of the crowd process design space. Specifically, the dimensions of this space are given by a set of the unique parameter definitions of all surface structures, including their nested crowd process fragments. This entails that all parameter declarations are mapped to a *hierarchical key* that uniquely identifies them. For example, the parameters for the Collection-fragment in Figure 14 (box highlighted in green), may be prefixed with their parent's key */FindAndFix/CollectDecide/Collection/*. Hence, each surface structure is represented by a data point in the optimizer's search space, whereas its coordinates are given by its (hierarchy-transformed) parameters as well as all other dimensions of the space set to 0.

4.4 (C) Running the optimizer

Once the optimizer is launched, it continuously samples the objective function at different locations by executing $\sigma(\varphi(V_i))$. In contrast to NAE, which iterated through all valid unique V_i's, the optimizer chooses the next V_i to sample by maximizing its acquisition function. It then composes and executes the corresponding crowd process $\varphi(V_i)$ using actual workers. The result is evaluated using σ and returned to the Bayesian Optimization Module (Spearmint [Snoek et al. 2012]). The choice for the next sampling point is done using the common *Expected Improvement* strategy with an *ARD Matérn 5/2 covariance Kernel*

in the Bayesian Optimization Module (please see [Snoek et al. 2012] for more details and a comparison between strategies). Figure 15 shows the interactions between the different modules of BOA. Note that BOA supports maximization and minimization of objective functions. In the following we designate maximization objective functions by μ and minimization objective functions by $-\mu$.

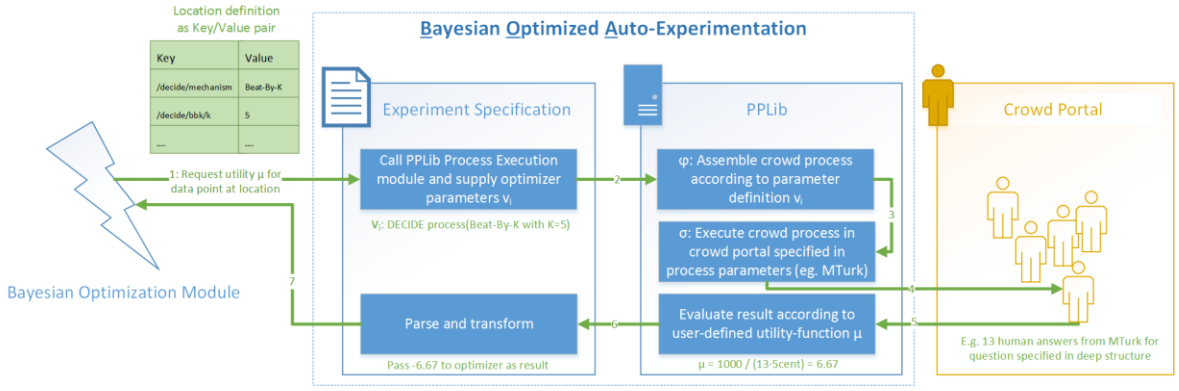

Figure 15: Workflow of (C) Running the optimizer with the use case of the optimizer requesting the utility for a crowd process using Beat-By-K (BBK) with K=5. The utility μ in the example is calculated by accuracy / cost, whereas accuracy of crowd answers is 1000

4.5 (D) Optimizer convergence

We stop the optimizer after it was unable to produce an increase in the highest observed utility during a predefined number of iterations. Experimentally, 20 seemed to offer a good tradeoff between cost and quality. After stopping, the process BOA estimated to reliably lead to the highest utility is returned. As our heuristic to find this reliably best performing process, BOA uses all samples of the objective function obtained by the optimizer to (i) determine its best encountered value and (ii) retrieve all sampled crowd process candidates with values within one standard deviation of this value. Depending on the user's preference between process stability vs quality, one could also include more (or less) process candidates. To avoid negative outliers, we calculate standard deviation across all samples that are better than the median sample. From this set, we (iii) choose the parameter combination V_i that exhibited the lowest utility variation across multiple executions giving preference to processes with multiple executions, where we assumed the distribution is sampled more precisely.

5 Evaluation

The central claims of this paper are, that crowd process selection can be efficiently solves through black-box optimization. Consequently, we need to show that solving this optimization problem does indeed lead to good solutions and that the Bayesian Optimization based BOA does so more efficiently (e.g. 10x less expensive) than the only black-box alternative known to us: NAE.

Hence, the following hypotheses need to be addressed in order (cited from page 15):

- H3.1: BOA nominates processes of equal or higher quality than have been proposed by experts before for a given problem
- H3.2: BOA finds good processes at a lower price than NAE

Before investigating these two hypotheses, it would be interesting to compare BOA with NAE directly. One of the main difficulties when evaluating crowd process performance is the lack of ground truth: Since crowd processes are inherently non-deterministic, using their repeated executions can only sample their true performance. This problem is aggravated by the large cost in obtaining samples from

crowd processes in a real-world setting. We therefore decided to run our comparison between BOA and NAE in a simulated setting, where we can compare thousands of processes (almost) for free, and where we can therefore approximate ground truth by re-executing each individual candidate process many times and using its median utility. This simulation allows us to compare BOA and NAE within the limitations of a simulated world. To ensure that these results generalize to the real world, and to investigate H3.1 and H3.2, the 2nd experiment is conducted within the context of a typical crowdsourcing task: text-shortening. For either hypothesis, the baseline is given by an expert-proposed solution, which in the context of text-shortening is given through the popular Find-Fix-Verify [Bernstein et al. 2010b]. To show that BOA scales to a large set of crowd process candidates, we ran a 3rd (and 4th) experiment, where we replicated the Bayesian Truth Serum setting [Prelec 2004] to a large extent. The Bayesian Truth Serum can be considered an expert-proposed solution to the corresponding crowdsourcing problem (further detailed below), which allows us to further investigate H3.1 and H3.2 in this larger real-world scenario.

Note, that in all evaluations, we specified our goals using a cost function $-\mu$ (to be minimized) instead of a utility function μ (to be maximized), since a cost function seemed more intuitive to explain the concepts. All real-world experiments were executed on Amazon Mechanical Turk, NAE was executed sequentially, compensation (if not stated differently) for multiple choice questions was 5 cents, free-text questions were priced at 15 cents. We only used US workers with less than 4% rejected HITs and more than 4000 approved HITs. All experiments were randomized and conducted on mornings of working days Eastern Time.

5.1 Experiment #1: BOA in a simulated environment

The goal of the simulation was to compare BOA and NAE in many repetitions to assess their mean performance and typical price difference. To this end, we simulated the task of asking a group of crowd workers a multiple-choice question with 4 possible answers and aggregating these to elicit the correct answer. This is a typical crowdsourcing pattern used, e.g., to tag images or in text sentiment analysis. The simulation consisted of four experimental conditions (C1-C4), corresponding to different levels of bias or uncertainty for the crowd choosing the correct answer. As shown in Table 7, each condition used a different set of probabilities for an answer to be chosen by a simulated crowd-worker, where the "correct" answer is always the one with the highest probability in a condition. Note that we have chosen the probabilities for a correct answer to be selected to decrease in each condition in favor of the others. This simulates higher levels of uncertainty about the correct answer. For example, in a multiple choice question with four options having the respective probabilities of 1%, 1%, 1% and 97% of being picked by a simulated crowd worker, an accordingly biased dice is rolled to decide which answer an individual agent will select.

	Answer 1	**Answer 2**	**Answer 3**	**Answer 4**
C1	1%	1%	1%	97%
C2	10%	10%	10%	70%
C3	20%	20%	20%	40%
C4	24%	24%	24%	28%

Table 7: Probabilities for each item to be picked in the different experimental conditions C1-C4

For all experimental conditions, we employed PPLib to identify the crowd process that reliably leads to the correct answer. Each condition employed one DECIDE-type building block of PPLib, which is used to elicit the answer to a multiple choice decision, as problem deep structure [de Boer and Bernstein 2016b]. Specifically, we set up PPLib to vary the following parameters:

- The coordination mechanism between crowd workers (Majority Vote, Beat-By-K [Goschin 2014], Confidence-based Voting [Barowy et al. 2012])
- The number of (simulated) crowd workers to use (between 1-10)
- K in Beat-By-K (between 1-10)
- The confidence value in Confidence-based Voting (between 0.65 and 0.95)
- The maximal amount of iterations in Beat-By-K and confidence-based voting (between 20 and 30)

The variation of these parameters on the DECIDE block lead PPLib's Recombinator to generate a set of 208 candidate crowd processes (or surface structures). For more information on recombination, please refer to the related work section and the PPLib article (see page 39).

Following the PPLib workflow, either a utility function or a cost function needs to be defined to enable Auto-Experimentation to identify the best suited crowd process from the candidates. We defined the cost function $-\mu$ to comply with our goal to favor answers closer to the "correct" one (i.e., the answer with the highest probability for being selected in the experimental condition) as:

$$-\mu(c_s, k_s) = \max_i(\pi(c_i)) - \pi(c_s) + k_s,$$

where the c_i's are the possible answers of the given experimental condition (see Table 7), c_s denotes the answer selected by the crowd process in a single execution s, k_s is the cost of execution s, and the function $\pi(c_i)$ returns the probability for the choice supplied in the argument (for example, when executing π with Answer 2 as an argument in C2, the result would be 10%). Hence, $-\mu$ calculates the difference between the highest result of π in an experiment condition $[\max_i(\pi(c_i))]$ and the probability of the item selected by the crowd $[\pi(c_s)]$ whilst adding the expense of the process k_s.

We ran each Auto-Experimentation algorithm (BOA, NAE1-5) 100 times for the four experimental conditions (C1-C4). The result's mean and standard deviation for both cost and utility of the 100 runs for each algorithm in condition C4 are shown in Table 8. We highlighted the relative cost of each algorithm when compared to BOA in red, emphasizing BOA's cost-efficiency. Indeed, BOA seems to be the least expensive algorithm, reliably finding well-performing crowd processes (as identified by their low average $-\mu$). NAE2-5 (where each crowd process gets executed 2-5 times, respectively) nominate slightly better processes on average; but at a much higher cost. NAE4-5 are omitted in Table 8, since they repeat NAE-3's performance on utility, whilst costing even more. The results of conditions C1-C3 follow the same trend.

		BOA	NAE-1	NAE-2	NAE-3
$-\mu$	Average	7.50	11.12	6.84	6.00
	Stdev	5.00	5.80	2.76	0.00
cost	Average	$8.18	$139.68	$279.34	$418.63
		1x	17x	34x	51x
	Stdev	$1.94	$2.31	$3.37	$3.88

Table 8: BOA vs NAE[1-3]'s utility and cost in C4 of the simulation.

When inspecting the differences in utility for all experimental conditions C1-C4, we find BOA to clearly outperform NAE-1. BOA does however get slightly outperformed by NAE-2.

We also found that each experimental condition (C1-C4) yielded different optimal processes, which emphasizes the need for systematic crowd process design and confirms a similar finding using real world-data [de Boer and Bernstein 2016b]. Conditions with a small difference in selection probabilities (C3, C4) are generally a lot more expensive than the conditions with high difference thereof (C1, C2). In addition to a higher cost in Auto-Experimentation, the nominated crowd processes are also more expensive. This makes sense intuitively: The more ambiguous the correct answer to a question, the lower the share of crowd workers answering correctly – therefore requiring more work to obtain the correct answer.

5.2 Experiment #2: BOA in the a real-world

Since an evaluation in a simulated environment is fundamentally limited by the assumptions taken to build the simulation, its generalizability to the real world is not always clear. To address this shortcoming, this section reports on a comparison between BOA and NAE in a real-world experiment. Moreover, a real-world experiment gives us the opportunity to test H3.1 and H3.2. Given its ubiquity in crowdsourcing, we chose the problem of text shortening, where crowd workers were employed to shorten a given article[35] from the popular news platform *The Verge* as much as possible without losing relevant information. The article had 1440 characters. An expert-design solution to the problem of text shortening is Find-Fix-Verify [Bernstein et al. 2010b]; in order to support H3.1, the nominated process in this evaluation would need to perform either equal or better than Find-Fix-Verify as measured by NAE. In order to support H3.2, it would need to be significantly less expensive than NAE.

We set the crowd-process's deep structure to a single CREATE-type building block (to *create* a shortened version of a given paragraph; see also [de Boer and Bernstein 2016b]) that operates on each paragraph of the original article. Since NAE entails running all generated crowd processes multiple times, it can become very costly. We therefore had to limit the Recombinator to vary only the following coordination mechanisms and associated parameters:

- Majority vote with 3 or 5 voters
- Beat-By-K with K=2
- Statistical Confidence Voting with 85% confidence
- Collections with 5 or 7 workers
- Sigma-pruned Collections with 5 or 7 workers

We used a trade-off between crowd process cost and the shortened text-version as utility function μ:

$$-\mu(l, c) = l + c,$$

where *l* is the resulting text-length and *c* is the crowd process execution cost in cents. Using this problem definition for text shortening, the Recombinator generated 41 different processes. For NAE-4 we could reuse experimental results acquired in [de Boer and Bernstein 2016b]. We configured BOA to sample (without replacement) from these.

The process nominated by BOA was a variation of Find-Fix-Verify [Bernstein et al. 2010a], using 3 crowd workers to propose shortened alternatives to a paragraph, and 5 crowd workers to select the best alternative in a majority vote. This finding supports H3.1. The nominated process was the 2nd best

[35]http://www.theverge.com/2015/1/8/7517361/google-getting-ready-to-sell-auto-insurance-and-maybe-buy-coverhound

of all evaluated processes as measured by $-\mu$ according to NAE-4, with a small delta μ of 12.3 (mean μ was 1643.6 with a standard deviation of 219.6). The total cost of running BOA was $44.40 as opposed to the $499.41 consumed by NAE-4, which supports H3.2.

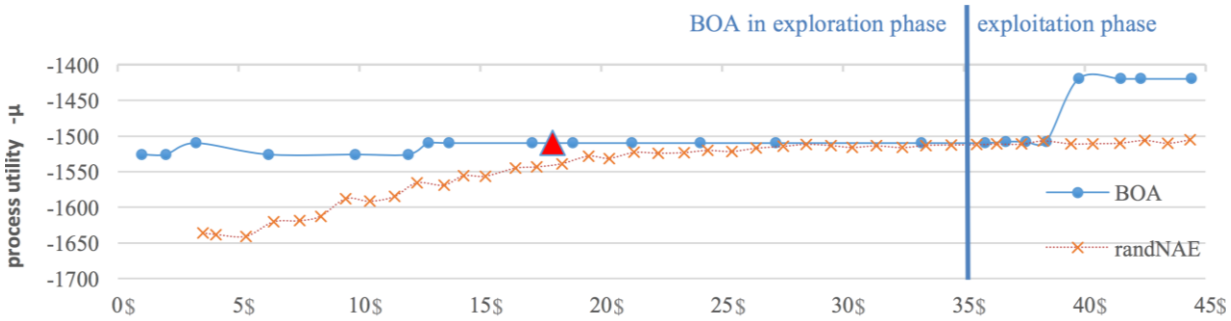

Figure 16: Average utility -μ of a strategy's nominated process (Y axis, higher is better) compared to money spent by that strategy (X-axis, in USD). Red triangle designates the first time BOA sampled its ultimately nominated process. For the same amount of money, BOA in average finds better processes than randNAE.

To illustrate the cost-efficiency of BOA, we modified NAE to randomly sample configurations without replacement rather than search the whole space. We call this setting randomized NAE (randNAE). We ran randNAE for budgets ranging from 1$ to BOA's convergence budget ($44.50), stopping it whenever it exceeded the budget in a given iteration (e.g., for a 1$ budget, the actual aggregate cost might be 3$, as one only knows it after running a process). This procedure was repeated 1000 times for each budget and the mean -μ and actual cost returned. As evidenced in Figure 16, BOA's result quality (Y-Axis) is higher in average than randNAE's at a given budget. The figure shows how BOA initially samples various locations of the process search space in what we call the exploration phase. During that phase, BOA primarily builds its knowledge about the search space and therefore appears to perform similar to random sampling. In this example, after having spent $35, BOA starts rechecking promising processes executed in the exploration phase and ultimately converges onto its final nomination. During this exploitation phase, one can see a clear difference between randNAE and BOA. BOA has encountered the process it would ultimately nominate in the middle of its exploration phase already (see red triangle in Figure 16). At that time, BOA was not confident enough to nominate it though, since another process had returned a better result when sampled once. BOA resampled it in the exploitation phase and found it to perform well and stable.

5.3 Experiment #3/4: BOA vs NAE for an expert process

The goal of the third experiment is to test whether H3 can be confirmed for a larger experimental setup, where NAE would be prohibitively expensive. Given that the literature provides a large catalogue of intricate crowd processes proposed by (academic) experts, we decided to replicate the experimental setup of an existing paper and investigate if BOA would nominate a crowd process with a comparable performance to the one proposed in the replicated paper.

Given its rich discussion and theoretical founding, we chose Prelec's Bayesian Truth Serum (BTS), formally introduced in [Prelec 2004]. The author verified his model in a recent field experiment involving human subjects [Prelec et al. 2014], where BTS is used to '*find truth even if the crowd is wrong*'. In this paper, we replicated the real-world study. BTS can be applied to a multiple choice question, where it essentially weights a person's choice by the accuracy of that person's estimate on how often each answer option to the question is selected by the other people polled. For example, when being

asked to pick the capital city of a US state among four cities, a person's answer contains her/his own reply and an assessment of that person's belief what fraction of the population would pick each of the four choices. In case of US states, some capitals are not among the four most populous cities of a given state, which may lead to wrong answers. For example, in our experiment, Philadelphia was often mistakenly selected as capital of Pennsylvania – even though the actual answer is Harrisburg. BTS is based on the assumption that people knowing the right answer to a question may be able to better predict what others confuse the right answer with and can therefore estimate the percentage of people giving that answer more accurately.

Since our main goal is to test the Auto-Experimentation module (not the Recombination mechanism), we extended PPLib's Process Repository with a new DECIDE-type building block implementing BTS. Furthermore, we have used the same questions as in the BTS paper: Determining the capital of US states among four cities, including at least three of the four most populous cities of that state besides the actual capital. The deep structure was, therefore, a simple DECIDE-type building block, which was executed on 10 randomly sampled US states (without replacement). We have limited the Recombinator to the same parameter ranges as in Experiment #1, but also included the new BTS building block and its variations on the amount of workers used (between 1-10) as alternatives to choose from. Our main goal was to find the crowd process leading to the most accurate labels. We therefore defined the cost function $-\mu$ as the number of misclassifications of the candidate crowd process, which is to be minimized.

The problem definition stated above lead the Recombinator to generate 212 candidate processes, including BTS. When executing BOA on the set of the 212 recombined candidate crowd processes, it was able to identify a process that performed even better in this task setting than Bayesian Truth Serum: A Beat-By-K voting pattern, with K=9, was nominated as the best crowd process (measured by process stability and result accuracy). Figure 17 shows each step of BOA in reaching this conclusion and displays the cost function values $-\mu$ of each executed crowd process candidate (Y-axis, lower is better) per step (X-axis). BOA took 21 steps to identify the winning process (by pure chance, it encountered the minimal -μ in the first step. Following our convergence criterion, the optimizer then stopped 20 iterations after being unable to produce a new minimal -μ). As the Figure indicates, it tested two variants of BTS (orange triangles; varying the number of involved crowd workers) and multiple versions of Beat-By-K (circles) eventually choosing Beat-By-K with K=9 as the winning process (red circle). Note that the run in step #3 aborted due to an internal error. In order to verify the cost aspect of H3, we need to determine the cost of executing NAE-4.

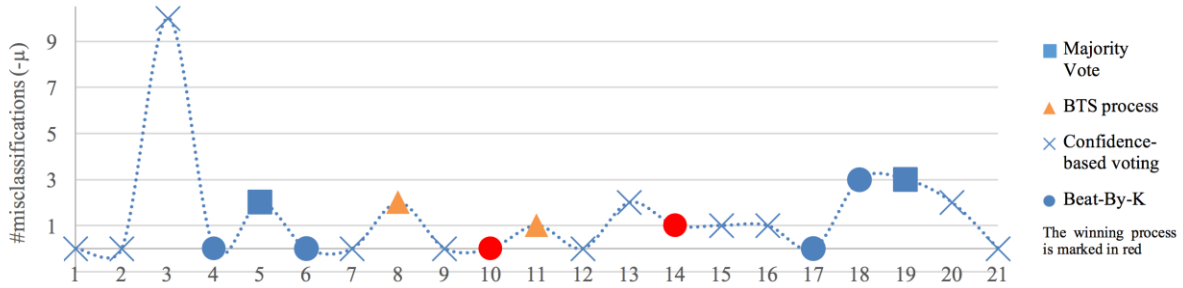

Figure 17: Results of cost functions $-\boldsymbol{\mu}$ of evaluated candidate crowd processes (Y-axis, lower is better) as experienced by BOA in each step (X-axis) when looking for the best crowd process in terms of accuracy. $-\boldsymbol{\mu}$ is the number of misclassifications. The shape of a data point designates the used control mechanism and captures any possible parameterization. Except for red shapes, which designate the winning variant of a control mechanism: Beat-By-K with K=9, while blue circles designate any other flavor.

Given the prohibitively expensive cost of executing NAE-4 for all 212 candidates, we chose to calculate a lower bound using the easily predictable cost of Majority Vote as a baseline. Majority vote is the least expensive pattern of all coordination structures in the set, as all other processes sample the same number of people, but then add some complications (such as requiring a difference of at least K, often

leading to additional votes required). NAE-4's total cost of evaluating all variations of a Majority Vote with worker counts between 1-10 is

$$□ \cdot □ \cdot \sum_{□=1}^{□} □ \cdot □ = 10 \cdot 4 \cdot \sum_{□=1}^{10} □ \cdot 4 = 88\$,$$

for 10 states (s), 4 NAE iterations (t), 4 cent per question (k), 1-10 workers (w). Since Beat-By-K includes 10 different parameter values for its maximal budget, its lower cost-bound is therefore ten times Majority Vote's. BTS' lower bound is five times Majority Votes, since it asks each crowd worker 5 questions in one task. Leaving Confidence-based Voting aside (as its cost is complicated to assess ex-ante), all of these four process account for a pessimistic total lower bound cost for NAE-4 of $88\$ + 10 \cdot 88\$ + 5 \cdot 88\$ = 1408\$$. BOA's actual cost amounts to less than 10% of that: \$96.24. Combined with the fact, that BTS was designed by an expert and that the nominated process was superior in performance, H3 is supported in this experimental setting as well.

Beat-By-K with K=9, seems to lead to very accurate results for finding US states capitals. It is, however, very expensive[36], since it requires the winning city to receive at least 9 more votes than the 2nd most popular city – and continues requesting more (costly) crowd votes until that condition is satisfied. Since our utility function only looked at accuracy, there was no punishment for Beat-By-K's high cost. Hence, we decided to rerun the experiment whilst incorporating crowd process cost into the utility as well:

$$-□(□, □) = □ \cdot 300 + □$$

where m is the number of misclassifications of a given crowd process and c the amount spent on running the process in cents. Loosely, each misclassified state capital lead to a punishment of 3\$ for the Auto-Experimentation Engine on top of the process' actual execution cost. Note that since the problem definition remains the same, we executed BOA on the same 212 generated crowd processes (only with a different cost function $-\mu$).

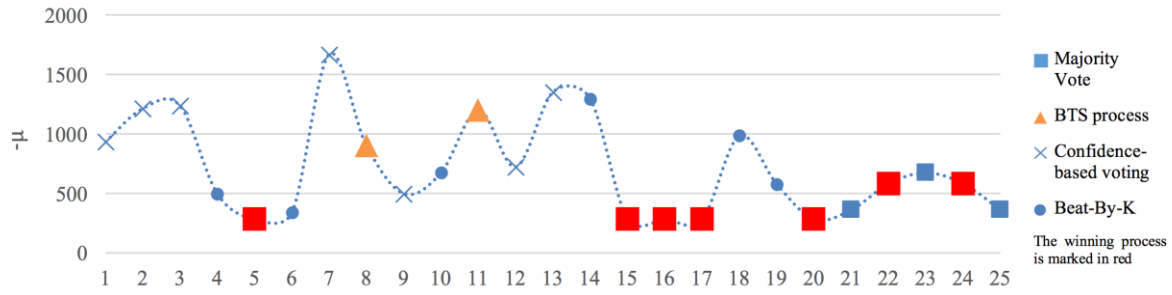

Figure 18: Values of cost functions $-\mu$ of evaluated candidate crowd processes (Y-axis, lower is better) as experienced by BOA in each step (X-axis) when optimizing for a trade-off between accuracy and cost. As in Figure 17, markers only show control structures

Figure 18 again shows each step taken by BOA in for this 2nd iteration. As expected, the cost based penalization of Beat-By-K-9 lead BOA to explore the tradeoff between accuracy and cost, nominating a simple majority vote with 7 crowd workers (red squares). The tested variations of the Bayesian Truth Serum (triangles) were among the most costly crowd processes, which significantly influenced their ranking: PPLib by default prices crowd tasks automatically according to the number of questions crowd workers need to answer in a single task (and therefore, the amount of time spent by a crowd worker to answer an individual crowd task). Since Bayesian Truth Serum relies on crowd workers estimating the probability of other crowd workers answering each individual city on top of her/his own choice, a single task was priced at 18 cents. While a single multiple choice question asking crowd workers to identify the capital city is only priced at 4 cents. BOA evaluated 2 variations of BTS, both leading to

[36] Its average cost was \$6.16 for 10 state capitals

rather high (i.e. unattractive) values of $-\mu$. In Figure 18, it is also interesting to observe that BOA repeatedly executes the winning process after step #15 (red squares) as it tries to assess its stability and then starts comparing it to other flavors of Majority Vote (blue squares).

In our 2nd iteration we again find BOA nominating a well-performing process below 10% of NAE's cost ($79.24). In order for the findings of this 2nd iteration to support H3.1 and H3.2, the nominated process would need to be equal or better than an expert-proposed one. Due to incorporating *cost* into the utility function, it is less clear whether BTS represents an expert-proposed crowd process for this particular use case. However, it is safe to say, that BTS is among the most relevant expert-proposed processes. Hence, we believe these results further support H3.1 and H3.2.

When comparing Figure 17 with Figure 18, another interesting observation is that the first 14 processes executed by BOA are the same in both runs. The reason is that the optimizer initially builds up its knowledge of the objective function by concurrently asking for 14 function evaluations and only then starts updating its priors for the next evaluation (at step 15). The number of concurrent initial data requests depends on the size of the set of the recombined surface structures, whereas a larger space leads to more initial observation requests to make reasonable judgments about where to sample next. Afterwards BOA progresses sequentially as it tries to minimize the spending for Auto-Experimentation by taking as much prior knowledge into account as possible. One could parallelize BOA more by sampling multiple points according to their acquisition value in each iteration. This would result in higher overall cost of BOA but faster execution. On average NAE would not incur differences in cost/quality when executed in parallel. Running the different variations of BOA in our evaluation took 10.1 hours in average (σ=3.6).

Reflecting upon these results, it is unintuitive that a Majority vote or Beat-By-K would be effective at '*finding the truth even if the majority is wrong*' – the task we set out to do originally by reusing the experiment setup of [Prelec et al. 2014]. Indeed, our collected data suggests that a slight majority of the crowd we used for our study (Mechanical Turkers) usually did point to the right state capital. Hence, it is important to point out how our study differs from the original study:

- Use of different crowd and different n: [Prelec et al. 2014] surveyed students at MIT (n=51) and Princeton (n=32), whilst we hired crowds on Mechanical Turk (n varies with every tested crowd process candidate between 1-10 as specified in the experiment setup)
- Different study design: The original study defines a Majority Vote as asking students whether the most populous city of a state is the state's capital (YES/NO), whilst our Majority Vote asked crowd workers to pick the capital among a list of the four most populous cities. For BTS, both studies supplied the same information to participants.
- Different number of classified states: The original study let all students specify the capitals of all US states whilst we randomly sampled 10 US states for cost reasons.
- Utility function: Our definition for the utility function μ (minimizing the number of misclassifications) might not capture the intent of the authors accurately.

The different environment of this study makes it not a reproduction of [Prelec et al. 2014]. Instead, we wanted BOA to nominate a fitting crowd process for the problem of state capital identification – which included comparing the arguably most popular approach proposed by an expert, BTS, with others. We believe, the difference in environments are sufficiently small to treat BTS as the current gold standard for state capital identification, which is a precondition of it being the baseline for testing our H3.

6 Limitations and future work

Given that BOA is a module of PPLib, it inherits some of PPLib's limitations such as the inability to explore crowd processes that cannot be assembled from PPLib building blocks [de Boer and Bernstein 2016b]. When focusing on BOA specifically, a few additional things need to be considered.

Most importantly, BOA is limited by the accuracy of the user's quantification of her (business-) objectives in the utility function μ. As the outcomes for the two different utility functions used in Experiment #3 clearly evidence, BOA's nomination is heavily dependent on μ. Defining the utility function accurately is therefore a paramount to successfully applying BOA in practice. We think, Judgment bootstrapping [Armstrong 2001] poses one path to an accurate μ. Please also note, that the utility functions used in our evaluation were defined by us according to our intuition of importance for a given problem. In our experiments, we only looked at deterministic utility functions μ, always returning the same result when supplied the same arguments.

The technique of Bayesian Optimization requires the error distribution of candidate crowd processes to be (almost) normal due to using GP's — a fact that was true in our simulation, but could not be guaranteed in any real-world experiments. However, the positive outcomes of the real-world experiments shed light on BOA's robustness for violations of its assumptions. Nonetheless, the robustness of BOA should be explored in the future.

Crowdsourcing markets and their characteristics may change over time. Therefore, one might want to correct for some confounding variables, such as day-of-week, for example by rerunning BOA on different days of the week or parameterizing execution time for BOA to optimize. Changes during BOA's execution are modelled as part of the crowd processes' variance and are hence accounted for.

We evaluated our approach to crowd process selection with up to 212 candidate processes to select from and saw that the optimizer found good processes relatively quickly. However, it is possible that our approach scales less well to (many) thousands of candidates than some methods that use insights about tasks to guide process selection [Dai et al. 2013; Zhang et al. 2013]. Future research should quantify the difference between these two approaches and possibly bridge the gap by introducing a hybrid between task-agnostic optimization, applicable to any crowd process, and task-based optimization, applicable only to crowd processes whose tasks performances have been modelled.

Running BOA (or any Auto-Experimentation strategy) is limited by the quality of a tasks sample, e.g., for the running example problem definition used in this paper of translating a book, a sample could be a set of paragraphs, pages or even chapters. The problem of finding representative samples is not limited to crowd process design though. Various approaches have been proposed to guide sampling, among them power analysis (with respect to sample size) [Cohen 1988].

There are many different ways of applying Bayesian Optimization to a problem. Further investigation is required to contrast different configurations for this problem. Instead of an optimization problem, crowd process selection could also be seen as a multi-armed bandit problem, where a gambler needs to decide on what machine to play, in which order to play them and how many times to play, whereas machines would correspond to crowd processes. Lin et al [Lin and Weld 2012] found that switching between multiple active crowd processes may yield better performance than using a single crowd process.

7 Conclusion

In this paper we proposed a strategy to efficiently select crowd processes from a potentially large set of candidates independent of these candidates' inner workings. When combining this technique with generative crowdsourcing libraries such as PPLib, a method to efficiently find well-performing crowd

processes for a problem at hand emerges. Our approach is based on the idea, that crowd processes can be composed of process fragments. These process fragments (and their individual configuration) can be seen as parameters to a black-box optimization problem, which we propose to solve using the Bayesian Optimization technique. We implemented our method as an extension to the PPLib system, an open source programming library to support crowd process designers. We evaluated our prototype by virtue of a simulation as well as three real-world experiments. The evaluation showed large cost-reductions when compared to the baseline algorithm for crowd process selection at a comparably high quality. Due to its black-box nature, our approach has a lower entry barrier for efficient crowd-process selection than current state-of-the-art. This allows practitioners to easily and affordably compare several candidate processes, which in turn has the potential to simplify the design of crowdsourcing solutions for many users.

8 Acknowledgements

We would like to thank Michael Feldman and Yiftach Nagar for their feedback on our first draft of this paper. This work was supported in part by the Swiss National Science Foundation (SNSF- Project: 200021- 143411/1).

Additionally, we'd like to express our gratitude to our anonymous reviewers and everybody else involved in the reviewing process, whose constructive feedback was extremely valuable in improving our paper.

Part III, Epilogue

Epilogue

Part II included three articles building up on each other; we first presented a concrete use case for crowd process design, where we reduced the complexity of an expert task in order to enable a diverse crowd to process it. This use case demonstrated the difficulty involved in crowd process design, as it took us more than a year to find it.

We then proposed a method and system capable of automatically deriving a set of crowd processes for any problem specification and selecting a fitting process from the set. Our final contribution was a methodology improving the efficiency of selecting a fitting crowd process among a set of candidates through Bayesian Optimization.

All of these contributions are subject to limitations. In the following, we will outline general limitations and directions for future work; specific limitations have been elaborated in their corresponding articles in Part II of this dissertation.

1 Limitations and Future Work

This sections outlines some of the most important limitations of the methods and results presented in this dissertation. More details for each individual project can be found in the respective sections in Part II of this thesis.

1.1 Generalizability

We could not test PPLib for all possible environments and crowdsourcing tasks. Instead, we tested it on a few real-world problems, of which we reported three in this dissertation. We additionally ran a simulation to test whether the crowd process selection mechanism could be reliable when scaling to thousands of executions. Reproduction by a third party as well as applications in additional real-world cases would shed more light on the validity of the hypotheses explored in this dissertation.

In similar vein, the crowd process created for the statistical review case study was only validated with papers of the HCI domain and, in a preliminary study, from the medical domain. Additional domains would need to be tested to gain further confidence in its generalizability.

1.2 Recombination

Our mechanism for recombination is based on generating all possible combinations of crowd process candidates, which leads to an exponentially growing search space. This stands in contrast to the limited funds and limited processing power available to us. In some cases, the search space might even be infinite, e.g. for continuous-valued parameters of crowd processes. Our current approach of separating *the generation of crowd process candidates* and *the selection* does not allow dealing with very large process search spaces. For these, it might be worthy to explore integrating recombination and selection, such that the Recombinator could be guided towards recombining useful processes based on Auto-Experimentation results or other heuristics.

Going one step further, a central authority could collect outputs of Auto-Experimentation Engines for different deployments of PPLib, ran by different users, on different problems. A dataset built this way might be useful to build a prediction model, that could unearth patterns for what crowd processes might work where, similar to how [Serban 2010b] outlined. This information could then be used to guide the Recombination process for a newly encountered problem. Essentially, this would relate Crowd Process Design for a new project to the cold start problem in recommender systems. [Serban 2010b] followed a similar approach in her dissertation.

Finally, the PPLib approach is limited by the abstract operators contained in the Process Repository. New design patterns and novel solutions to specific problems need to be incorporated into the Process Repository for Recombination to work adequately. PPLib, the system, is built to allow for decentralized extension of the Process Repository through standard dependency managers for modern programming languages.

1.3 Auto-Experimentation

As outlined in the description of Research Question Q3, we let the user define utility functions for a given problem. The definition of these utility functions through a mathematical formula is not trivial to humans if multiple factors are involved to calculate crowd process fitness. A possible remedy in practice might be Judgment Bootstrapping [Armstrong 2001]: inferring the weights of the utility function through regression, based on ratings for target values. Multi-Objective Optimization avoids the definition of weights for individual objectives and instead focuses on the pareto front established by them. Points on the pareto front are optimal with respect to the trade-off between the multiple objectives: Improving one of the objectives further comes at the expense of the others. This works as long as all objectives are equally important to the user. There are various sorts of Multi-Objective Optimization, some of them (e.g. [Davins-Valldaura et al. 2016]) might satisfy the criteria outlined in RQ3 for crowd process selection and could be worthy of trying in the future. The main difficulty seems to lie with minimizing the number of executions of the fitness function for cost reasons.

Our use of Bayesian Optimization has been tested as a means to nominate a fitting process among roughly 200 candidate crowd processes. Since we use a black-box approach to optimization, scaling to many *thousands* of candidates would likely aggravate the curse of dimensionality[37] and therefore decrease the efficiency of black-box approaches. To support this case, a hybrid approach that incorporates some knowledge about individual subtasks of a workflow (e.g. [Zhang et al. 2013]) might be worth exploring.

2 Conclusion

With the increasing adoption of crowdsourcing in practice and academia, the question of *how to coordinate crowd workers to solve a given task* has become ever more important. Borrowing from the notion of business processes in organizational design, different variations of crowd processes may be used for a given problem; but the design of such a crowd process can currently be considered more of an art than science. A possible remedy are crowd design patterns, such as Find-Fix-Verify or Iterative Refinement. However, their performance for a given problem is unknown before actually implementing and applying them; and as we have found in this thesis, performance may vary greatly between different problems.

The first article of this thesis is devoted to a case study for crowd process design, where we describe and evaluate a crowd process for an expert task: The reviewing of statistical reporting of research articles. We found our crowdsourcing-based approach to perform slightly less well than actual expert's, but coming at very low cost and the capability of scaling to a large number of articles. This allowed us to use it to investigate reporting standards of a top-tier HCI conference over time. We found, that most authors do not report whether they have checked statistical assumption before applying statistical methods, and that the reporting standards did not improve significantly over time.

[37] Increasing dimensionality leads to such a strong increase of space, that available data becomes much sparser. The curse of dimensionality is a common problem in (dynamic-) optimization.

Working on this case study for crowd process design convinced us of the need for tools and methods supporting crowd process designers. In the second article of this thesis, we therefore propose PPLib, a method and system that automatically derives crowd processes for given problem using a repository of existing crowd process fragments. Based on a problem description as abstract operators, these crowd process fragments can be recombined to derive a set of candidate processes for a given problem. The candidates are then executed on a sample problem (e.g. when translating a book, a sample could be translating one page) and rated by a user-defined utility function to establish their performance. We validate PPLib in two real-world problems, and find it to derive crowd processes yielding adequate results for either.

A weakness of this version of PPLib is, that all candidate processes need to be executed in order to select one satisfying a user's performance requirements; this can quickly become expensive. In the third article, we therefore propose to use black-box optimization to more efficiently identify a process satisfying a user's requirements. We implement our solution as a module to the open source PPLib system and evaluate it in two real-world experiments and one simulation. Our results indicate, that optimization-based crowd process selection nominates suitable processes at lower cost than the previous baseline.

Combined, the PPLib approach proposed in this dissertation demonstrates the advantages of automated crowd process design and hereby helps pave the way for wider adoption of crowdsourcing in practice.

References

Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. 2011. The Jabberwocky Programming Environment for Structured Social Computing. *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), 53. DOI:http://dx.doi.org/10.1145/2047196.2047203

Leona Aiken et al. 1999. Statistical Methods in Psychology Journals. 54, 8 (1999), 594–604. DOI:http://dx.doi.org/10.1037/0003-066X.54.8.594

Peter J. Allen et al. 2016. Introducing StatHand: A Cross-Platform Mobile Application to Support Students' Statistical Decision Making. *Frontiers in psychology* 7, February (2016), 288. DOI:http://dx.doi.org/10.3389/fpsyg.2016.00288

Jürgen Angele, Michael Kifer, and Georg Lausen. 2009. Ontologies in F-Logic. In *Handbook on Ontologies*. 45–70. DOI:http://dx.doi.org/10.1007/978-3-540-92673-3_2

Jon Scott Armstrong. 2001. Judgmental bootstrapping: Inferring experts' rules for forecasting. *Principles of forecasting: A Handbook for Researchers and Practitioners* (2001), 169–192.

Monya Baker. 2016. Statisticians issue warning on Pvalues. *Nature* 531 (2016), 151. DOI:http://dx.doi.org/10.1038/nature.2016.19503

Daniel Barowy, Charlie Curtsinger, Emery Berger, and Andrew McGregor. 2012. AutoMan: A platform for integrating human-based and digital computation. *OOPSLA '12 Proceedings of the ACM international conference on Object oriented programming systems languages and applications* (2012). DOI:http://dx.doi.org/10.1145/2384616.2384663

Steve Battle et al. 2005. Semantic web services language (SWSL). *W3C Member Submission 9* (2005).

James Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. 2011. Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems* (2011), 2546–2554.

Abraham Bernstein. 2000. How can cooperative work tools support dynamic group process? Bridging the specificity frontier. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (CSCW '00)*. 279–288. DOI:http://dx.doi.org/10.1145/358916.358999

Abraham Bernstein, Mark Klein, and Thomas W. Malone. 2012. Programming the global brain. *Communications of the ACM* 55 (2012), 41. DOI:http://dx.doi.org/10.1145/2160718.2160731

Abraham Bernstein, Mark Klein, and Thomas W. Malone. 1999. The Process Recombinator: A Tool for Generating New Business Process Ideas. In *ICIS '99 Proceedings of the 20th international conference on Information Systems*. Atlanta, GA, USA: Association for Information Systems, 178–192.

Abraham Bernstein, Foster Provost, and Shawndra Hill. 2005. Towards Intelligent Assistance for a Data Mining Process: An Ontology-based Approach for Cost-sensitive Classification. *IEEE Transactions on Knowledge and Data Engineering* 17 (2005), 503–518. DOI:http://dx.doi.org/10.1109/TKDE.2005.67

Michael S. Bernstein et al. 2010a. Soylent : A Word Processor with a Crowd Inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. 313–322.

Michael S. Bernstein et al. 2010b. Soylent : A Word Processor with a Crowd Inside. In *UIST*.

William D. Berry and Stanley Feldman. 1985. *Multiple regression in practice*,

Patrick de Boer and Abraham Bernstein. 2017. Efficiently Identifying a Well-Performing Crowd Process for a Given Problem. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (2017), 1688–1699. DOI:http://dx.doi.org/10.1145/2998181.2998263

Patrick M. de Boer and Abraham Bernstein. 2016a. Efficient exploration of the crowd process design space. In *Collective Intelligence*. 1–4.

Patrick M. de Boer and Abraham Bernstein. 2015. PPLib : Towards Systematic Crowd Process Design

using Recombination and Auto-Experimentation. In *Collective Intelligence*. 1–4.

Patrick M. de Boer and Abraham Bernstein. 2016b. PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation. *ACM Transactions on Intelligent Systems and Technology* , Special Issue: Crowd in Intelligent Systems (2016).

Eric Brochu, Vlad Cora, and Nando De Freitas. 2010. *A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning*,

Justin Cheng and Michael S. Bernstein. 2015. Flock : Hybrid Crowd-Machine Learning Classifiers. *CSCW 2015* (2015), 600–611. DOI:http://dx.doi.org/10.1145/2675133.2675214

Noam Chomsky. 1965. Aspects of the Theory of Syntax. *MIT Press* (1965). DOI:http://dx.doi.org/10.1016/0732-118X(86)90008-5

Jacob Cohen. 1988. Statistical power analysis for the behavioral sciences. *Statistical Power Analysis for the Behavioral Sciences* 2nd (1988), 567. DOI:http://dx.doi.org/10.1234/12345678

Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. 2013. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence* 202 (2013), 52–85. DOI:http://dx.doi.org/10.1016/j.artint.2013.06.002

Joan Davins-Valldaura, Saïd Moussaoui, Guillermo Pita-Gil, and Franck Plestan. 2016. ParEGO extensions for multi-objective optimization of expensive evaluation functions. *Journal of Global Optimization* 67, 1 (2016), 1–18. DOI:http://dx.doi.org/10.1007/s10898-016-0419-3

Robyn M. Dawes. 1971. A case study of graduate admissions: Application of three principles of human decision making. *American Psychologist* 26, 2 (1971), 180–188. DOI:http://dx.doi.org/10.1037/h0030868

Pierre Dragicevic. 2016. *Fair Statistical Communication in HCI*,

Stephen E. Edgell and Sheila M. Noon. 1984. Effect of violation of normality on the t test of the correlation coefficient. *Psychological Bulletin* 95, 3 (1984), 576–583. DOI:http://dx.doi.org/10.1037/0033-2909.95.3.576

Eyda Ertekin, Haym Hirsh, and Cynthia Rudin. 2012. Selective Sampling of Labelers for Approximating the Crowd. In *2012 AAAI Fall Symposium Series*. 7–13.

Andy Field. 2013. Discovering Statistics using IBM SPSS Statistics. *Discovering Statistics using IBM SPSS Statistics* (2013), 297–321. DOI:http://dx.doi.org/10.1016/B978-012691360-6/50012-4

Lorenz Fischer, Shen Gao, and Abraham Bernstein. 2015. Machines Tuning Machines: Configuring Distributed Stream Processors with Bayesian Optimization. *2015 IEEE International Conference on Cluster Computing* (2015), 22–31. DOI:http://dx.doi.org/10.1109/CLUSTER.2015.13

Martin Fowler. 2004. Inversion of Control Containers and the Dependency Injection pattern. *http://martinfowler.com/articles/injection.html* (2004).

Michael Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: answering queries with crowdsourcing. *SIGMOD '11: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (2011), 1–12. DOI:http://dx.doi.org/10.1145/1989323.1989331

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. Automated Planning, theory and practice. *Elsevier* (2004).

Phillip I. Good and James W. Hardin. 2012. *Common errors in statistics (and how to avoid them)*, John Wiley & Sons.

Sergiu Goschin. 2014. *Stochastic dilemmas: foundations and applications*. Rutgers University-Graduate School-New Brunswick.

M. Hammer and J. Champy. 1993. Business process re-engineering. *London: Nicholas Brealey* (1993).

Michael Hammer, Gerry Howe, Vincent Kruskal, and Irving Wladawsky. 1977. A Very High Level Language for Data Processing Applications. *Communications of the ACM* 20, 11 (1977), 832–840.

Rink Hoekstra, Henk A.L. Kiers, and Addie Johnson. 2012. Are assumptions of well-known statistical techniques checked, and why (not)? *Frontiers in Psychology* 3, MAY (2012), 1–9. DOI:http://dx.doi.org/10.3389/fpsyg.2012.00137

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6683 LNCS (2011), 507–523. DOI:http://dx.doi.org/10.1007/978-3-642-25566-3_40

Oana Inel, Khalid Khamkham, Tatiana Cristea, and Anca Dumitrache. 2014. CrowdTruth : Machine-Human Computation Framework for Harnessing Disagreement in Gathering Annotated Data. In *International Semantic Web Conference (ISWC)*.

Joshua Introne, Robert Laubacher, Gary Olson, and Thomas Malone. 2011. The Climate CoLab: Large scale model-based collaborative planning. In *Proceedings of the 2011 International Conference on Collaboration Technologies and Systems, CTS 2011*. 40–47. DOI:http://dx.doi.org/10.1109/CTS.2011.5928663

Panos Ipeirotis. 2010. *Demographics of Mechanical Turk*,

Jeffrey Kahn. 2011. Reporting statistics in APA style. *Geraadpleegd op* (2011).

Maurits Kaptein and Judy Robertson. 2012. Rethinking Statistical Analysis Methods for CHI. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), 1105–1114. DOI:http://dx.doi.org/10.1145/2207676.2208557

Matthew Kay, Gregory L. Nelson, and Eric B. Hekler. 2016. Researcher-Centered Design of Statistics. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16* , August (2016), 4521–4532. DOI:http://dx.doi.org/10.1145/2858036.2858465

Peter Kincaid, Robert Fishburne Jr, Richard Rogers, and Brad Chissom. 1975. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. *Naval Technical Training Command Millington TN Research Branch* (1975).

Ross D. King et al. 2009. The automation of science. *Science (New York, N.Y.)* 324 (2009), 85–89. DOI:http://dx.doi.org/10.1126/science.1165620

A. Kittur, J. Nickerson, and M. Bernstein. 2013. The Future of Crowd Work. *Proc. CSCW '13* (2013), 1–17. DOI:http://dx.doi.org/10.1145/2441776.2441923

Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. 2011. CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. 43–52. DOI:http://dx.doi.org/10.1145/2047196.2047202

David Kocsis and Gert-Jan de Vreede. 2016. Towards a Taxonomy of Ethical Considerations in Crowdsourcing. *AMCIS* (2016), 10.

Anand Kulkarni, Matthew Can, and Björn Hartmann. 2012. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*. 1003. DOI:http://dx.doi.org/10.1145/2145204.2145354

William Kuzon, Melanie G. Urbanchek, and Steven James Mccabe. 1997. Seven deadly sins of statistical analysis. *Journal of Oral and Maxillofacial Surgery* 55, 8 (1997), 897–898. DOI:http://dx.doi.org/10.1016/S0278-2391(97)90377-3

Jintae Lee, George M. Wyner, and Brian T. Pentland. 2008. Process grammar as a tool for business process design. *MIS Quarterly* 32 (2008), 757–778. DOI:http://dx.doi.org/Article

Jeffrey T. Leek and Roger D. Peng. 2015. P values are just the tip of the iceberg. *Nature* 520, 7549 (2015),

612. DOI:http://dx.doi.org/10.1038/520612a

Beatrice Liem and Yiling Chen. 2011. An Iterative Dual Pathway Structure for Speech-to-Text Transcription. In 37–42.

Ch Lin and Ds Weld. 2012. Dynamically Switching between Synergistic Workflows for Crowdsourcing. In *26th Conference on Artificial Intelligence, AAAI*. 132–133.

Chris J. Lintott et al. 2008. Galaxy Zoo: Morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society* 389, 3 (2008), 1179–1189. DOI:http://dx.doi.org/10.1111/j.1365-2966.2008.13689.x

Greg Little. 2010. Exploring iterative and parallel human computation processes. *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems CHI EA 10* (2010), 4309. DOI:http://dx.doi.org/10.1145/1753846.1754145

Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. 2010. TurKit: Human Computation Algorithms on Mechanical Turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*. 57. DOI:http://dx.doi.org/10.1145/1866029.1866040

Benjamin Livshits and Todd Mytkowicz. 2014. Saving Money While Polling with InterPoll Using Power Analysis. In *Conference on Human Computation and Crowdsourcing*. 159–170.

Thomas W. Malone. 1987. Modeling Coordination in Organizations and Markets. *Management Science* 33, 10 (1987), 1317–1332. DOI:http://dx.doi.org/10.1287/mnsc.33.10.1317

Thomas W. Malone et al. 1999. Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science* 45, 3 (1999), 425–443.

Thomas W. Malone et al. 2003. Tools for inventing organizations: Toward a handbook of organizational processes. In Thomas W. Malone, Kevin Crowston, & George Herman, eds. *Organizing Business Knowledge: The MIT Process Handbook*. Cambridge, MA: MIT Press.

Thomas W. Malone and Kevin Crowston. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys* 26 (1994), 87–119. DOI:http://dx.doi.org/http://doi.acm.org/10.1145/174666.174668

Thomas W. Malone, Kevin Crowston, and George A. Herman. 2003. *Organizing Business Knowledge: the MIT Process Handbook*,

Thomas W. Malone, Robert Laubacher, and Chrysanthos Dellarocas. 2010. The collective intelligence genome. *IEEE Engineering Management Review* 38 (2010), 38. DOI:http://dx.doi.org/10.1109/EMR.2010.5559142

Thomas W. Malone, Robert J. Laubacher, and Tammy Johns. 2011. The big idea: The age of hyperspecialization. *Harvard Business Review* 89 (2011), 2.

Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. 2015. Bayesian Optimization for Contextual Policy Search. (2015), 1–2.

Katrina Panovich Michael S. Bernstein, Greg Little, Robert C. Miller, Bjöïrn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell. 2015. Soylent: A Word Processor with a Crowd Inside. *Communications of the ACM* 58, 8 (2015), 85–94.

Patrick Minder and Abraham Bernstein. 2012. CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems. In Karl Aberer, Andreas Flache, Wander Jager, Ling Liu, Jie Tang, & Christophe Guéret, eds. *Social Informatics*. Springer Berlin Heidelberg, 124–137. DOI:http://dx.doi.org/10.1007/978-3-642-35386-4_10

Jonas Mockus. 2012. *Bayesian approach to global optimization: theory and applications* Vol. 37. S., Kluwer Academic Publishers.

Yiftach Nagar, Patrick De Boer, and Ana Cristina Bicharra Garcia. 2016. Accelerating the Review of Complex Intellectual Artifacts in Crowdsourced Innovation Challenges. *Proceedings of the Thirty*

Seventh International Conference on Information Systems (ICIS2016) (2016), 1–17.

Allen Newell and Herbert Alexander Simon. 1972. *Human problem solving*, Englewood Cliffs, N.J.,: Prentice-Hall.

Michèle B. Nuijten, Chris H.J. Hartgerink, Marcel A.L.M. van Assen, Sacha Epskamp, and Jelte M. Wicherts. 2015. The prevalence of statistical reporting errors in psychology (1985–2013). *Behavior Research Methods* , 2011 (2015), 1–22. DOI:http://dx.doi.org/10.3758/s13428-015-0664-2

Regina Nuzzo. 2014. Statistical errors: P values, the "gold standard" of statistical validity, are not as reliable as many scientists assume. *Nature* 506, 7487 (2014), 150–152. DOI:http://dx.doi.org/10.1136/bmj.1.6053.66

Jason Osborne and Elaine Waters. 2002. Four assumptions of multiple regression that researchers should always test. *Practical Assessment, Research and Evaluation* 8, 2 (2002), 1. DOI:http://dx.doi.org/http://pareonline.net/getvn.asp?v=8&n=2

Pawel Popiel. 2017. "Boundaryless" in the creative economy: Assessing freelancing on Upwork. *Critical Studies in Media Communication* 0, 0 (2017), 1–14. DOI:http://dx.doi.org/10.1080/15295036.2017.1282618

Drazen Prelec. 2004. A Bayesian truth serum for subjective data. *Science* 306, 5695 (2004), 462–466. DOI:http://dx.doi.org/10.1126/science.1102081

Drazen Prelec, H. Sebastian Seung, and John McCoy. 2014. Finding truth even if the crowd is wrong. *Working paper* MIT (2014), 1–13.

Moo-ryong Ra, Bin Liu, Tom F. La Porta, and Ramesh Govindan. 2012. Medusa: a programming framework for crowd-sensing applications. *Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12* , Section 2 (2012), 337. DOI:http://dx.doi.org/10.1145/2307636.2307668

Jonas Ranstam. 2012. Why the P-value culture is bad and confidence intervals a better alternative. *Osteoarthritis and Cartilage* 20, 8 (2012), 805–808. DOI:http://dx.doi.org/10.1016/j.joca.2012.04.001

Jeffrey N. Rouder, Richard D. Morey, Josine Verhagen, Jordan M. Province, and Eric-Jan Wagenmakers. 2016. Is There A Free Lunch In Inference? *topiCS* 8, 1 (2016), 1–5. DOI:http://dx.doi.org/10.1007/s13398-014-0173-7.2

Maytal Saar-Tsechansky and Foster Provost. 2001. Active learning for class probability estimation and ranking. *IJCAI International Joint Conference on Artificial Intelligence* (2001), 911–917.

Niloufar Salehi, Andrew McCabe, Melissa Valentine, and Michael S. Bernstein. 2016. Huddler: Convening Stable and Familiar Crowd Teams Despite Unpredictable Availability. *Proceedings of the 20th ACM Conference on Computer Supported Cooperative Work & Social Computing* (2016). DOI:http://dx.doi.org/10.1145/2998181.2998300

Paul Van Schaik and Matthew Weston. 2016. Magnitude-based inference and its application in user research. *International Journal of Human Computer Studies* 88, August (2016), 38–50. DOI:http://dx.doi.org/10.1016/j.ijhcs.2016.01.002

John R. Searle. 1969. *Speech acts: an essay in the philosophy of language*, London,: Cambridge U.P.

Floarea Serban. 2010a. Auto-experimentation of KDD workflows based on ontological planning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 313–320. DOI:http://dx.doi.org/10.1007/978-3-642-17749-1_22

Floarea Serban. 2010b. Auto-experimentation of KDD workflows based on ontological planning. In *Lecture Notes in Computer Science*. 313–320. DOI:http://dx.doi.org/10.1007/978-3-642-17749-1_22

Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. 2013. A Survey of Intelligent Assistants for Data Analysis. *ACM Comput. Surv.* 45 (2013), 31:1–31:35. DOI:http://dx.doi.org/10.1145/2480741.2480748

Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 2008. Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 614. DOI:http://dx.doi.org/10.1145/1401890.1401965

Herbert Alexander Simon. 1977. *The new science of management decision* Rev., Englewood Cliffs, N.J.: Prentice-Hall.

Jasper Snoek, Hugo Larochelle, and Rp Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. *Nips* (2012), 1–9. DOI:http://dx.doi.org/2012arXiv1206.2944S

A.B.L. Srivastava. 2016. Biometrika Trust Effect of Non-Normality on the Power Function of t-Test. 45, 3 (2016), 421–430.

Alexander Strasak, Qamruz Zaman, Karl Pfeiffer, Georg Gobel, and Hanno Ulmer. 2007. Statistical errors in medical research a review of common pitfalls. *Swiss Med WKLY* 137 (2007), 44–49. DOI:http://dx.doi.org/2007/03/smw-11587

James Surowiecki. 2005. The Wisdom of Crowds. *American Journal of Physics* 75 (2005), 336. DOI:http://dx.doi.org/10.1038/climate.2009.73

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2013), 847–855. DOI:http://dx.doi.org/10.1145/2487575.2487629

Stefano Tranquillini, Florian Daniel, and Pavel Kucherbaev. 2015. Modeling , Enacting , and Integrating Custom Crowdsourcing Processes. *ACM Transactions on the Web (TWEB)* 9, 2 (2015), 1–43.

Karl T. Ulrich and Steven D. Eppinger. 1995. Product Design and Development. *Tata McGraw-Hill Education* (1995).

Andras Vargha and Harold Delaney. 1998. The Kruskal-Wallis Test and Stochastic Homogeneity. *Journal of Educational and Behavioral Statistics* 23, 2 (1998), 170–192. DOI:http://dx.doi.org/10.3102/10769986023002170

Coosje L.S. Veldkamp, Michele B. Nuijten, Linda Dominguez-Alvarez, Marcel A.L.M. Van Assen, and Jelte M. Wicherts. 2014. Statistical reporting errors and collaboration on statistical analyses in psychological science. *PLoS ONE* 9, 12 (2014), 1–19. DOI:http://dx.doi.org/10.1371/journal.pone.0114876

Vasilis Verroios and Michael S. Bernstein. 2014. Context Trees: Crowdsourcing Global Understanding from Local Views. In *HCOMP 2014*. Stanford InfoLab.

Chat Wacharamanotham, Krishna Subramanian, Sarah Theres Völkel, and Jan Borchers. 2015. Statsplorer: Guiding Novices in Statistical Analysis. *Proceedings of the ACM CHI'15 Conference on Human Factors in Computing Systems* 1 (2015), 2693–2702. DOI:http://dx.doi.org/10.1145/2702123.2702347

Daniel S. Weld, Mausam, and Peng Dai. 2011. *Human Intelligence Needs Artificial Intelligence*,

Jacob Westfall and Tal Yarkoni. 2016. Statistically controlling for confounding constructs is harder than you think. *PloS one* 11, 3 (2016), 1–22. DOI:http://dx.doi.org/10.1017/CBO9781107415324.004

Christopher J. Wild. 2000. Chance encounters: A first course in data analysis and inference. (2000).

Terry Winograd and Fernando Flores. 1986. *Understanding Computers and Cognition: a New Foundation for Design*, Norwood, N.J.: Ablex Pub. Corp.

Niklaus Wirth. 1971. Program Development by Stepwise Refinement. *Communications of the ACM* 14, 4 (1971), 221–227.

George Wyner and Jintae Lee. 2002. Process specialization: defining specialization for state diagrams. *Computational & Mathematical Organization Theory* 8 (2002), 133–155.

DOI:http://dx.doi.org/10.1023/A:1016091900743

Haoqi Zhang, Eric Horvitz, and D. Parkes. 2013. Automated workflow synthesis. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence* (2013), 1020–1026.

Haiyi Zhu, Steven P. Dow, Robert E. Kraut, and Aniket Kittur. 2014. Reviewing versus doing. In *Proceedings of the 17th ACM conference on Computer supported cooperative work - CSCW '14*. 1445–1455. DOI:http://dx.doi.org/10.1145/2531602.2531718

Donald W. Zimmerman. 2004. Inflation of Type I error rates by unequal variances associated with parametric, nonparametric, and rank-transformation tests. *Psicologica* 25, 1 (2004), 103–133.

Michael D. Zisman. 1978. Office Automation: Revolution or Evolution? *Sloan Management Review* 19, 3 (1978), 1–16.

University of Zurich[UZH]

Department of Informatics

University of Zurich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zurich
Phone +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch

Bibliographic citations of the papers constituting this thesis

Paper 1: Assessing Statistical Assumption Reporting in CHI

At the time of writing, this paper has been submitted to Conference on Human Factors in Computing Systems

Paper 2: PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation

This paper is published as *de Boer Patrick, Bernstein Abraham. "PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation" ACM TIST,* 7(4), *2016*

Paper 3: Efficiently Identifying a Well-Performing Crowd Process for a Given Problem

This paper is published as *de Boer Patrick, Bernstein Abraham. "Efficient Exploration of the Crowd Process Design Space", CSCW, 1688-1699, 2017*